Student : Esteban Ordenes

Post Graduate Program in Data Science and Business Analytics

PGP-DSBA-UTA-Dec20-A

CreditCard Users Churn Prediction

Context

The Thera bank recently saw a steep decline in the number of users of their credit card, credit cards are a good source of income for banks because of different kinds of fees charged by the banks like annual fees, balance transfer fees, and cash advance fees, late payment fees, foreign transaction fees, and others. Some fees are charged to every user irrespective of usage, while others are charged under specified circumstances.

Customers' leaving credit cards services would lead bank to loss, so the bank wants to analyze the data of customers and identify the customers who will leave their credit card services and reason for same – so that bank could improve upon those areas

Objective

Thera bank need to come up with a classification model that will help the bank improve their services so that customers do not renounce their credit cards.

- Explore and visualize the dataset.
- Build a classification model to predict if the customer is going to churn or not
- Optimize the model using appropriate techniques
- Generate a set of insights and recommendations that will help the bank

Data Dictionary

- CLIENTNUM: Client number. Unique identifier for the customer holding the account
- Attrition_Flag: Internal event (customer activity) variable - if the account is closed then 1 else 0
- Customer_Age: Age in Years
- Gender: Gender of the account holder
- Dependent_count: Number of dependents
- Education_Level: Educational Qualification of the account holder
- Marital_Status: Marital Status of the account holder
- Income_Category: Annual Income Category of the account holder
- Card_Category: Type of Card
- Months_on_book: Period of relationship with the bank
- Total_Relationship_Count: Total no. of products held by the customer
- Months_Inactive_12_mon: No. of months inactive in the last 12 months
- Contacts_Count_12_mon: No. of Contacts in the last 12 months

- Credit_Limit: Credit Limit on the Credit Card
- Total_Revolving_Bal: The balance that carries over from one month to the next is the revolving balance
- Avg_Open_To_Buy: Open to Buy refers to the amount left on the credit card to use (Average of last 12 months)
- Total_Trans_Amt: Total Transaction Amount (Last 12 months)
- Total_Trans_Ct: Total Transaction Count (Last 12 months)
- Total_Ct_Chng_Q4_Q1: Ratio of the total transaction count in 4th quarter and the total transaction count in 1st quarter
- Total_Amt_Chng_Q4_Q1: Ratio of the total transaction amount in 4th quarter and the total transaction amount in 1st quarter
- Avg_Utilization_Ratio: Represents how much of the available credit the customer spent

Loading libraries

In [511…

```
import warnings
warnings.filterwarnings("ignore")

# Libraries to help with reading and manipulating data

import pandas as pd
import numpy as np
import scipy.stats as stats

# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# To impute missing values
from sklearn.impute import KNNImputer

# To build a logistic regression model
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
# To tune model, get different metric scores and split data
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.metrics import accuracy_score,precision_score,recall_score
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
#from sklearn.metrics import accuracy_score, recall_score, precision_score, roc_auc_sco

from sklearn.pipeline import Pipeline, make_pipeline

#libraries to help with model building
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingRegressor, AdaBoostRegressor, StackingRegr
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
```

```
# Removes the limit from the number of displayed columns and rows.
# This is so I can see the entire dataframe when I  print it
pd.set_option("display.max_columns", None)
# pd.set_option('display.max_rows', None)
pd.set_option("display.max_rows", 200)

# To build linear model for statistical analysis and prediction
import statsmodels.stats.api as sms
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
from statsmodels.tools.tools import add_constant

# For pandas profiling
#from pandas_profiling import ProfileReport


import warnings
warnings.filterwarnings('ignore')
```

Read the dataset

In [512…
```
Loan = pd.read_csv("BankChurners.csv")
```

In [513…
```
# copying data to another variable to avoid any changes to original data
data = Loan.copy()
```

View the first and last 5 rows of the dataset.

In [514…
```
data.head()
```

Out[514…

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status
0	768805383	Existing Customer	45	M	3	High School	Married
1	818770008	Existing Customer	49	F	5	Graduate	Single
2	713982108	Existing Customer	51	M	3	Graduate	Married
3	769911858	Existing Customer	40	F	4	High School	Unknown
4	709106358	Existing Customer	40	M	3	Uneducated	Married

In [515…
```
data.tail()
```

Out[515…

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_S
10122	772366833	Existing Customer	50	M	2	Graduate	

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_S
10123	710638233	Attrited Customer	41	M	2	Unknown	Div
10124	716506083	Attrited Customer	44	F	1	High School	M
10125	717406983	Attrited Customer	30	M	2	Graduate	Unk
10126	714337233	Attrited Customer	43	F	2	Graduate	M

Understand the shape of the dataset.

In [516…
```
data.shape
```

Out[516…
```
(10127, 21)
```

- The dataset has 10127 rows and 21 columns

Check data types and number of non-null values for each column.

In [517…
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   CLIENTNUM                 10127 non-null  int64  
 1   Attrition_Flag            10127 non-null  object 
 2   Customer_Age              10127 non-null  int64  
 3   Gender                    10127 non-null  object 
 4   Dependent_count           10127 non-null  int64  
 5   Education_Level           10127 non-null  object 
 6   Marital_Status            10127 non-null  object 
 7   Income_Category           10127 non-null  object 
 8   Card_Category             10127 non-null  object 
 9   Months_on_book            10127 non-null  int64  
 10  Total_Relationship_Count  10127 non-null  int64  
 11  Months_Inactive_12_mon    10127 non-null  int64  
 12  Contacts_Count_12_mon     10127 non-null  int64  
 13  Credit_Limit              10127 non-null  float64
 14  Total_Revolving_Bal       10127 non-null  int64  
 15  Avg_Open_To_Buy           10127 non-null  float64
 16  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 17  Total_Trans_Amt           10127 non-null  int64  
 18  Total_Trans_Ct            10127 non-null  int64  
 19  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 20  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: float64(5), int64(10), object(6)
memory usage: 1.6+ MB
```

- We can see that there are total 21 columns and 10127 number of rows in the dataset.
- All columns' data type is either integer, float or object.
- There are NO null values in the columns. We can further confirm this using isna() method.

In [518…
```
data.isna().sum()
```

Out[518…
```
CLIENTNUM                 0
Attrition_Flag            0
Customer_Age              0
Gender                    0
Dependent_count           0
Education_Level           0
Marital_Status            0
Income_Category           0
Card_Category             0
Months_on_book            0
Total_Relationship_Count  0
Months_Inactive_12_mon    0
Contacts_Count_12_mon     0
Credit_Limit              0
Total_Revolving_Bal       0
Avg_Open_To_Buy           0
Total_Amt_Chng_Q4_Q1      0
Total_Trans_Amt           0
Total_Trans_Ct            0
Total_Ct_Chng_Q4_Q1       0
Avg_Utilization_Ratio     0
dtype: int64
```

Summary of the dataset

In [519…
```
# Summary of continuous columns
data[[ 'Credit_Limit' , 'Avg_Open_To_Buy' , 'Total_Amt_Chng_Q4_Q1' , 'Total_Ct_Chng_Q4_
```

Out[519…

	count	mean	std	min	25%	50%	75%	r
Credit_Limit	10127.0	8631.953698	9088.776650	1438.3	2555.000	4549.000	11067.500	34516.
Avg_Open_To_Buy	10127.0	7469.139637	9090.685324	3.0	1324.500	3474.000	9859.000	34516.
Total_Amt_Chng_Q4_Q1	10127.0	0.759941	0.219207	0.0	0.631	0.736	0.859	3.
Total_Ct_Chng_Q4_Q1	10127.0	0.712222	0.238086	0.0	0.582	0.702	0.818	3.
Avg_Utilization_Ratio	10127.0	0.274894	0.275691	0.0	0.023	0.176	0.503	0.

- `Credit_Limit` mean and median is 8631.95 and 4549 respectively.
- `Avg_Open_To_Buy` mean and median 7469.14 and 3474 respectively.
- `Total_Amt_Chng_Q4_Q1` mean and median is 0.759 and 0.736 respectively.
- `Total_Ct_Chng_Q4_Q1` mean and median is 0.712 and 0.702 respectively,
- `Avg_Utilization_Ratio` mean and median is 0.274 and 0.176 respectively.

Converting the data type of categorical features to 'category'

In [520…
```
cat_cols = ['Attrition_Flag'
            , 'Gender'
            , 'Education_Level'
            , 'Marital_Status'
            , 'Income_Category'
            , 'Card_Category']
```

```
data[cat_cols] = data[cat_cols].astype('category')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype   
---  ------                    --------------  -----   
 0   CLIENTNUM                 10127 non-null  int64   
 1   Attrition_Flag            10127 non-null  category
 2   Customer_Age              10127 non-null  int64   
 3   Gender                    10127 non-null  category
 4   Dependent_count           10127 non-null  int64   
 5   Education_Level           10127 non-null  category
 6   Marital_Status            10127 non-null  category
 7   Income_Category           10127 non-null  category
 8   Card_Category             10127 non-null  category
 9   Months_on_book            10127 non-null  int64   
 10  Total_Relationship_Count  10127 non-null  int64   
 11  Months_Inactive_12_mon    10127 non-null  int64   
 12  Contacts_Count_12_mon     10127 non-null  int64   
 13  Credit_Limit              10127 non-null  float64 
 14  Total_Revolving_Bal       10127 non-null  int64   
 15  Avg_Open_To_Buy           10127 non-null  float64 
 16  Total_Amt_Chng_Q4_Q1      10127 non-null  float64 
 17  Total_Trans_Amt           10127 non-null  int64   
 18  Total_Trans_Ct            10127 non-null  int64   
 19  Total_Ct_Chng_Q4_Q1       10127 non-null  float64 
 20  Avg_Utilization_Ratio     10127 non-null  float64 
dtypes: category(6), float64(5), int64(10)
memory usage: 1.2 MB
```

In [521…

```
# Summary of categorical columns
data[['Attrition_Flag'
            , 'Gender'
            , 'Education_Level'
            , 'Marital_Status'
            , 'Income_Category'
            , 'Card_Category']].describe(include='category').T
```

Out[521…

	count	unique	top	freq
Attrition_Flag	10127	2	Existing Customer	8500
Gender	10127	2	F	5358
Education_Level	10127	7	Graduate	3128
Marital_Status	10127	4	Married	4687
Income_Category	10127	6	Less than $40K	3561
Card_Category	10127	4	Blue	9436

In [522…

```
# inspect discrete columns
print('\n\nAttrition_Flag')
print(data.Attrition_Flag.value_counts())
print('\n\nGender')
print(data.Gender.value_counts())
print('\n\nEducation_Level')
print(data.Education_Level.value_counts())
print('\n\nMarital_Status')
print(data.Marital_Status.value_counts())
```

```
print('\n\nIncome_Category')
print(data.Income_Category.value_counts())
print('\n\nCard_Category')
print(data.Card_Category.value_counts())

print('\n\nDependent_count')
print(data.Dependent_count.value_counts())
```

```
Attrition_Flag
Existing Customer    8500
Attrited Customer    1627
Name: Attrition_Flag, dtype: int64


Gender
F    5358
M    4769
Name: Gender, dtype: int64


Education_Level
Graduate         3128
High School      2013
Unknown          1519
Uneducated       1487
College          1013
Post-Graduate     516
Doctorate         451
Name: Education_Level, dtype: int64


Marital_Status
Married     4687
Single      3943
Unknown      749
Divorced     748
Name: Marital_Status, dtype: int64


Income_Category
Less than $40K    3561
$40K - $60K       1790
$80K - $120K      1535
$60K - $80K       1402
Unknown           1112
$120K +            727
Name: Income_Category, dtype: int64


Card_Category
Blue        9436
Silver       555
Gold         116
Platinum      20
Name: Card_Category, dtype: int64


Dependent_count
3    2732
2    2655
1    1838
4    1574
0     904
```

```
5     424
Name: Dependent_count, dtype: int64
```

- `Attrition_Flag` is our target variable
- Existing Customer 8500 (86%)
- Attrited Customer 1627 (16%)

In [523…
```
# check for unique values for each column
data.nunique()
```

Out[523…
```
CLIENTNUM                 10127
Attrition_Flag                2
Customer_Age                 45
Gender                        2
Dependent_count               6
Education_Level               7
Marital_Status                4
Income_Category               6
Card_Category                 4
Months_on_book               44
Total_Relationship_Count      6
Months_Inactive_12_mon        7
Contacts_Count_12_mon         7
Credit_Limit               6205
Total_Revolving_Bal        1974
Avg_Open_To_Buy            6813
Total_Amt_Chng_Q4_Q1       1158
Total_Trans_Amt            5033
Total_Trans_Ct              126
Total_Ct_Chng_Q4_Q1         830
Avg_Utilization_Ratio       964
dtype: int64
```

- We can drop 'CustomerID' column as it is an ID variable and will not add value to the model.

In [524…
```
#Dropping CLIENTNUM columns from the dataframe, since this is a unique identifier and d
data.drop(columns=['CLIENTNUM'], inplace=True)
```

Lets Evaluate the Dependant Variable - Attrition_Flag

In [525…
```
data['Attrition_Flag'].value_counts()
```

Out[525…
```
Existing Customer    8500
Attrited Customer    1627
Name: Attrition_Flag, dtype: int64
```

- 16% of the customers are Attrited Customer
- Since we need to evaluate the Attrition_Flag variable, it makes more sense to convert this variable into Boolean where:
  - Existing Customer = 0 (false)
  - Attrited Customer = 1 (true)

In [526…
```
def convertAttritionFlagToBool(attritionValue):
    if attritionValue == 'Attrited Customer' :
```

```
        return True
    else:
        return False
```

In [527…
```
data['Attrition_Flag'] = data['Attrition_Flag'].apply(convertAttritionFlagToBool)
```

In [528…
```
data['Attrition_Flag'].value_counts()
```

Out[528…
```
False    8500
True     1627
Name: Attrition_Flag, dtype: int64
```

In [ ]:
```
```

EDA

Univariate analysis

In [529…
```
def histogram_boxplot(feature , figsize=(15,10) , bins=None):
    """ Histogram and Boxplot combined
    feature: 1-d feature array
    figsize: size of figg.default (15,10)
    bins: number of bins.default None/auto
    """
    mean = feature.mean()
    median = feature.median()
    mode = feature.mode()

    f2, (ax_box2 , ax_hist2) = plt.subplots(nrows = 2, # num of rows of the subplot. gr
                                            sharex = True, # x-axis will be shared amon
                                            gridspec_kw = { "height_ratios": (.25 , .75
                                            figsize = figsize
                                           ) # create the 2 subplots

    sns.boxplot(feature , ax = ax_box2 , showmeans = True , color = 'red') # boxplot wi
    if bins:
        sns.distplot(feature , kde = True , ax = ax_hist2, bins = bins)
    else:
        sns.distplot( feature , kde = True , ax = ax_hist2 )
    ax_hist2.axvline( mean , color = 'green' , linestyle='-' , linewidth = 3 , label =
    ax_hist2.axvline( median , color = 'yellow' , linestyle='-' , linewidth = 6 , label
    ax_hist2.axvline( mode[0] , color = 'black' , linestyle='-' , label = 'mode' ) # ad
    ax_hist2.legend()

    print( 'Mean:'+ str( mean ) )
    print( 'Median:'+ str( median ) )
    print( 'Mode:'+ str( mode[0] ) )
```

In [530…
```
def bar_count_pct( feature , figsize=(10,7) ):
    """
    feature : 1-d categorical feature array
    """
    mode = feature.mode()
    freq = feature.value_counts().max()

    #if isinstance(feature , int):
    #    cnt = feature.unique()
```

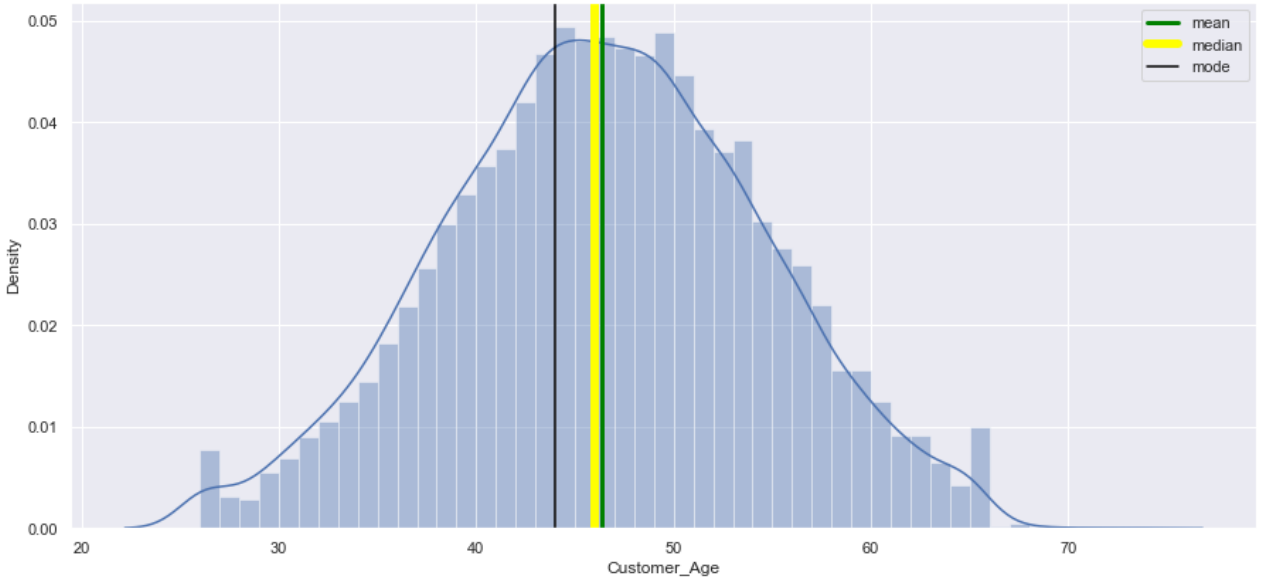
```
    #else:
    #    cnt  = feature.unique().value_counts().sum()

    plt.figure(figsize=figsize)

    ax = sns.countplot(feature)

    total = len(feature) # length of the column
    for p in ax.patches:
        percentage = '{:.1f}%'.format( 100 * p.get_height() / total ) # percentage of e
        x = p.get_x() + p.get_width() / 2 - 0.05 # width of the plot
        y = p.get_y() + p.get_height() # height of the plot
        ax.annotate( percentage , (x,y), size = 12) # annotate the percentage

    print( 'Top:'+ str( mode[0] ) )
    print( 'Freq:'+ str( freq ) )
```

Observation on Customer_Age

In [531…

```
histogram_boxplot(data.Customer_Age)
```

```
Mean:46.32596030413745
Median:46.0
Mode:44
```



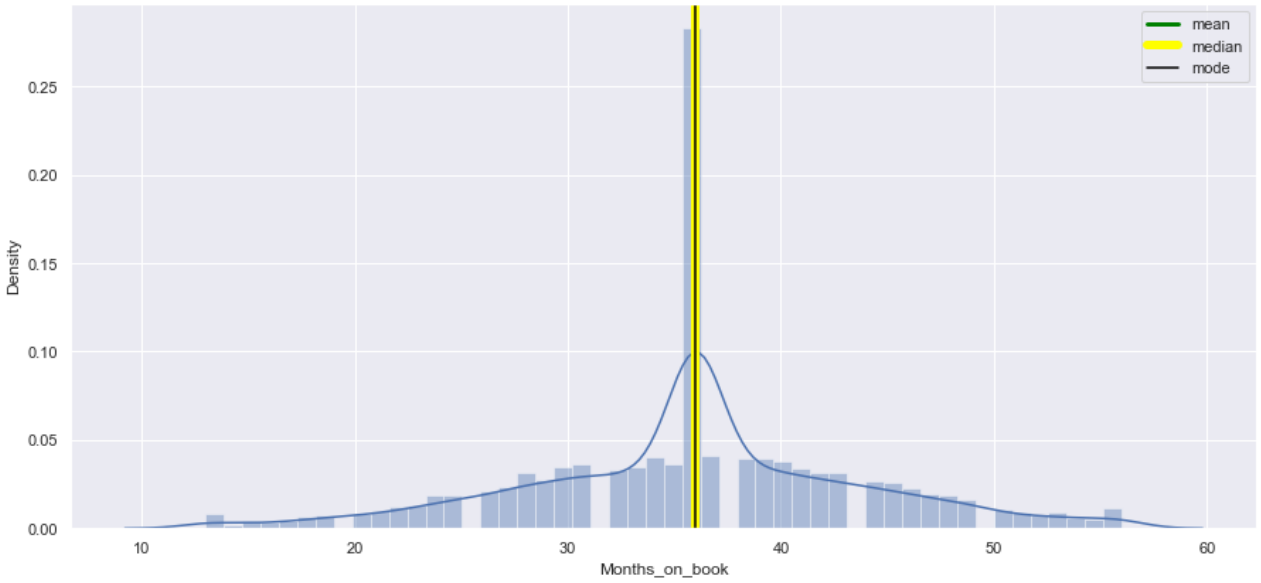
- Customer_Age feature is normally distributed.
- There are only a few outliers to the right of the curve.

Observation on Months_on_book

In [532…

```
histogram_boxplot(data.Months_on_book)
```

```
Mean:35.928409203120374
Median:36.0
Mode:36
```



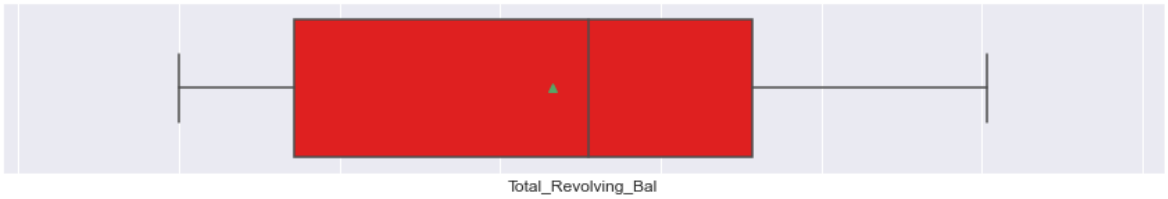
- Months_on_book feature is normally distributed.
- There are only a few outliers to the right and left of the curve.

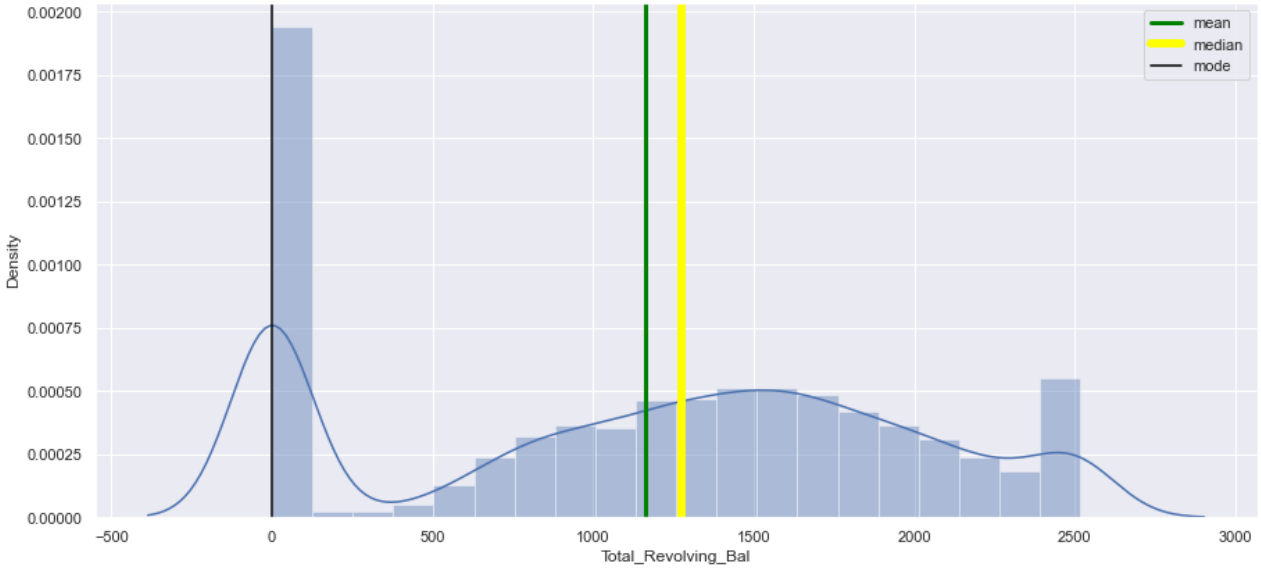
Observation on Total_Revolving_Bal

In [533…

```
histogram_boxplot(data.Total_Revolving_Bal  )
```

```
Mean:1162.8140614199665
Median:1276.0
Mode:0
```





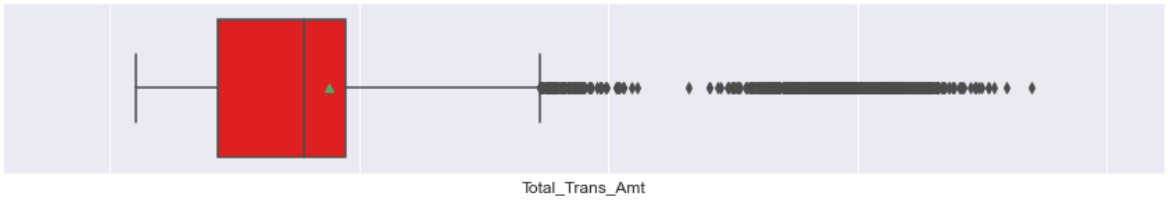
- Total_Revolving_Bal feature is Bi-modal distributed.
- This could be read as Total_Revolving_Bal being ditributed into 2 group, those that have 0 Total_Revolving_Bal and those that do not.
- There are no outliers, but large volume of the data point are at the extremes of the distribution.

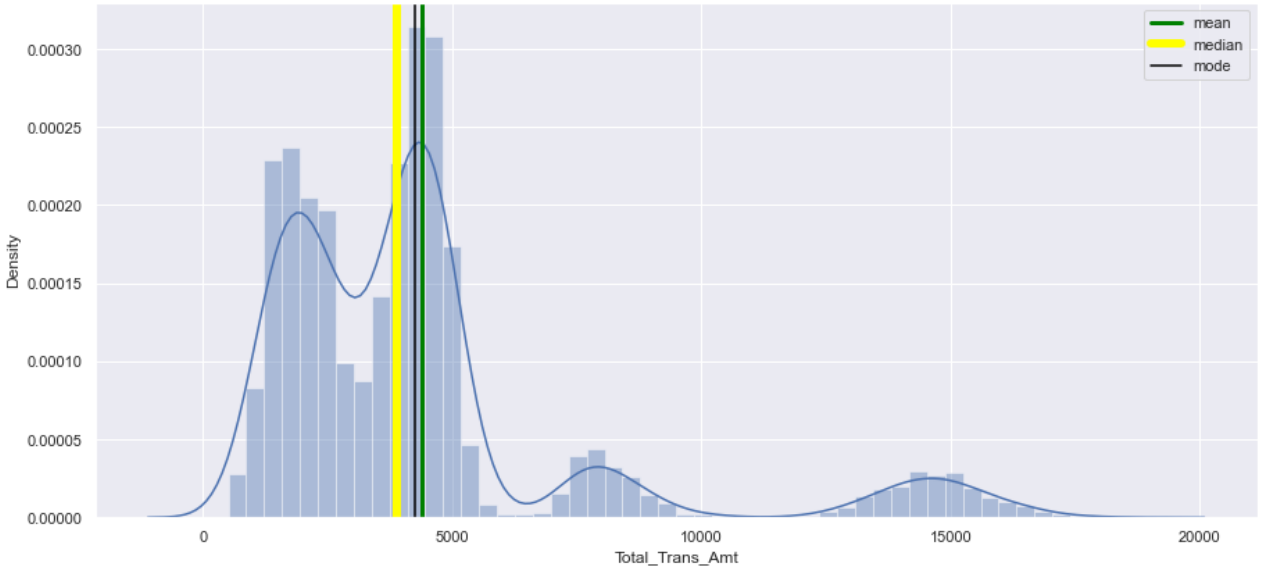
Observation on Total_Trans_Amt

In [534…

```
histogram_boxplot(data.Total_Trans_Amt)
```

```
Mean:4404.086303939963
Median:3899.0
Mode:4253
```





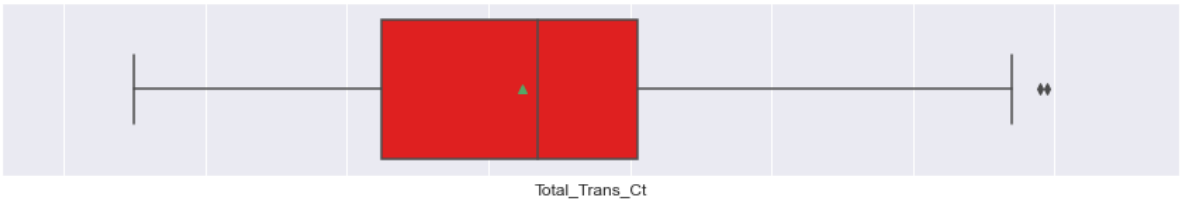
- Total_Trans_Amt feature is right-skewed.
- Total_Trans_Amt feature is Multi-modal distributed.
- This could be read as Total_Trans_Amt being ditributed into 4 groups.
- There are many outliers to the right of the curve which may explain its skewness.

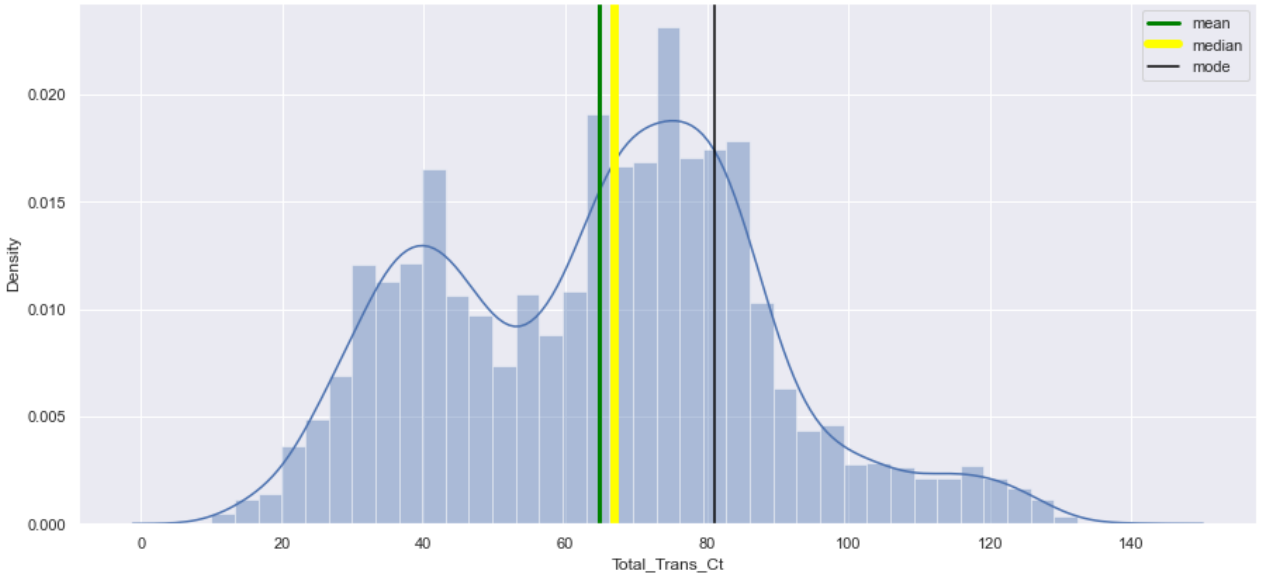
Observation on Total_Trans_Ct

In [535…

```
histogram_boxplot(data.Total_Trans_Ct)
```

```
Mean:64.85869457884863
Median:67.0
Mode:81
```





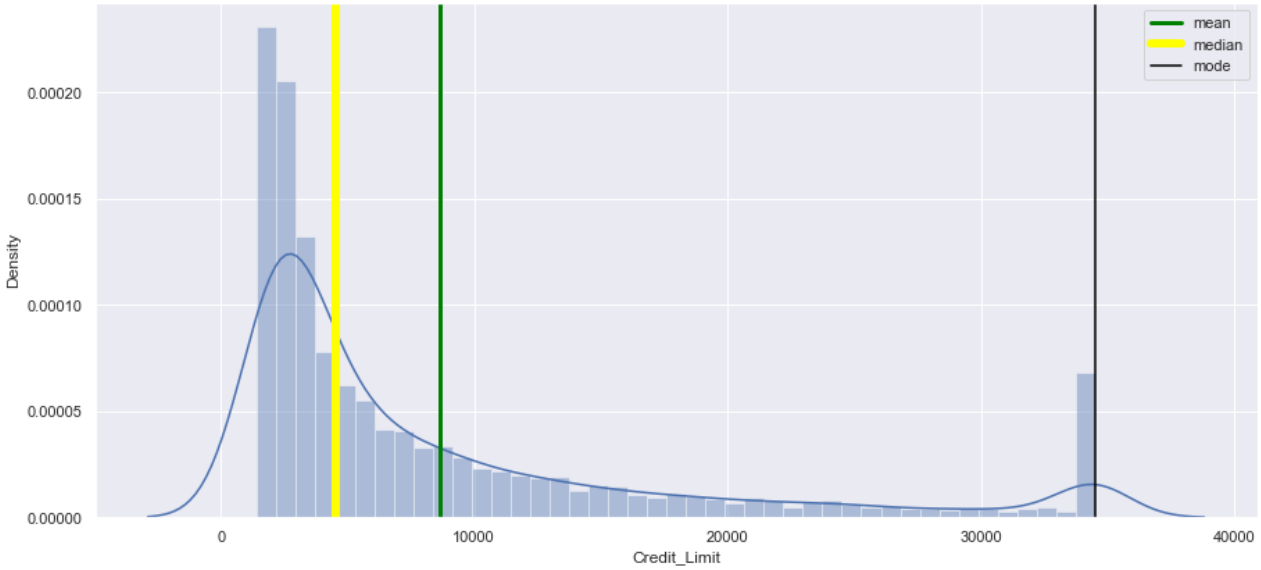
- Total_Trans_Ct feature is Bi-modal distributed.
- This could be read as Total_Trans_Ct being ditributed into 2 group, those that have Total_Trans_Ct below 55 and those above 55.
- There are few outliers.

Observation on Credit_Limit

In [536…

```
histogram_boxplot(data.Credit_Limit              )
```

```
Mean:8631.953698034848
Median:4549.0
Mode:34516.0
```



- Credit_Limit feature is right skewed.
- There are many outliers to the right of the distribution which could explain its skewness.

Observation on Avg_Open_To_Buy

In [537…

```
histogram_boxplot(data.Avg_Open_To_Buy)
```

```
Mean:7469.139636614887
Median:3474.0
Mode:1438.3
```

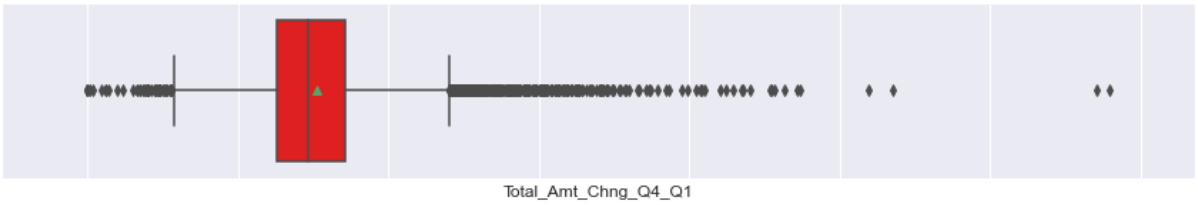
- Avg_Open_To_Buy feature is right skewed.
- There are many outliers to the right of the distribution which could explain its skewness.

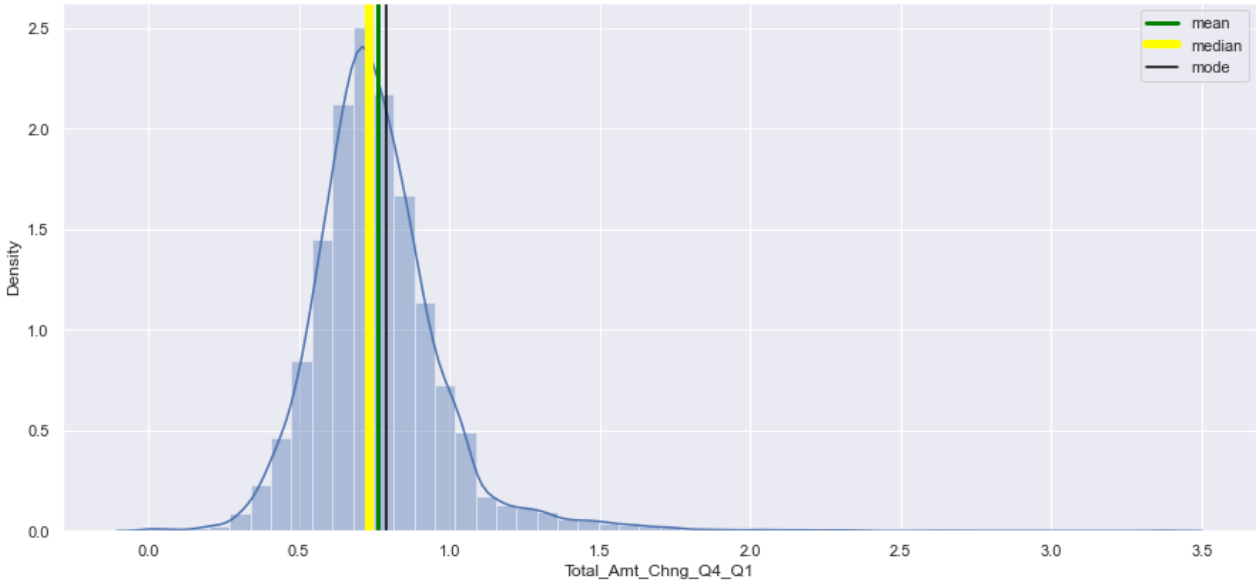
Observation on Total_Amt_Chng_Q4_Q1

In [538…

```
histogram_boxplot(data.Total_Amt_Chng_Q4_Q1)
```

```
Mean:0.7599406536980376
Median:0.736
Mode:0.7909999999999999
```





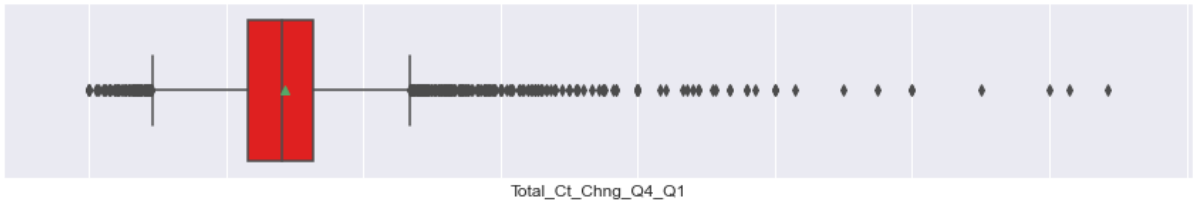
- Total_Amt_Chng_Q4_Q1 feature is right skewed.
- There are many outliers to the right and left of the distribution.

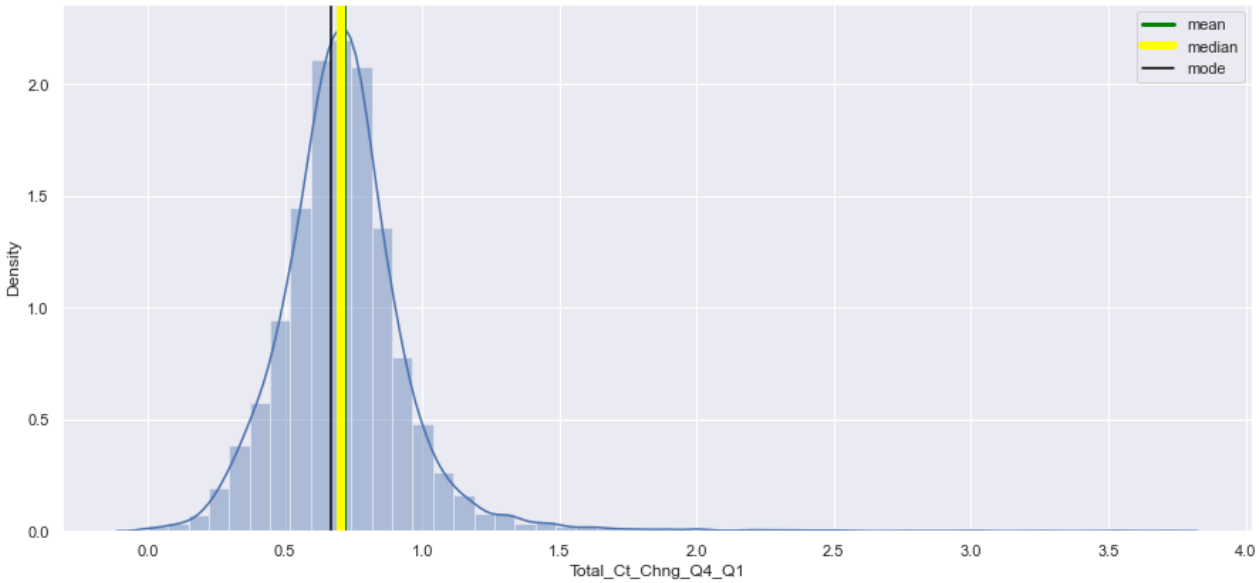
Observation on Total_Ct_Chng_Q4_Q1

In [539…

```
histogram_boxplot(data.Total_Ct_Chng_Q4_Q1)
```

```
Mean:0.7122223758269962
Median:0.7020000000000001
Mode:0.667
```





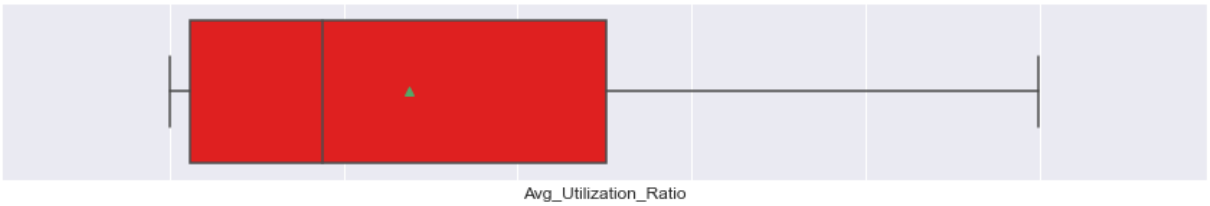
- Total_Ct_Chng_Q4_Q1 feature is right skewed.
- There are many outliers to the right and left of the distribution.

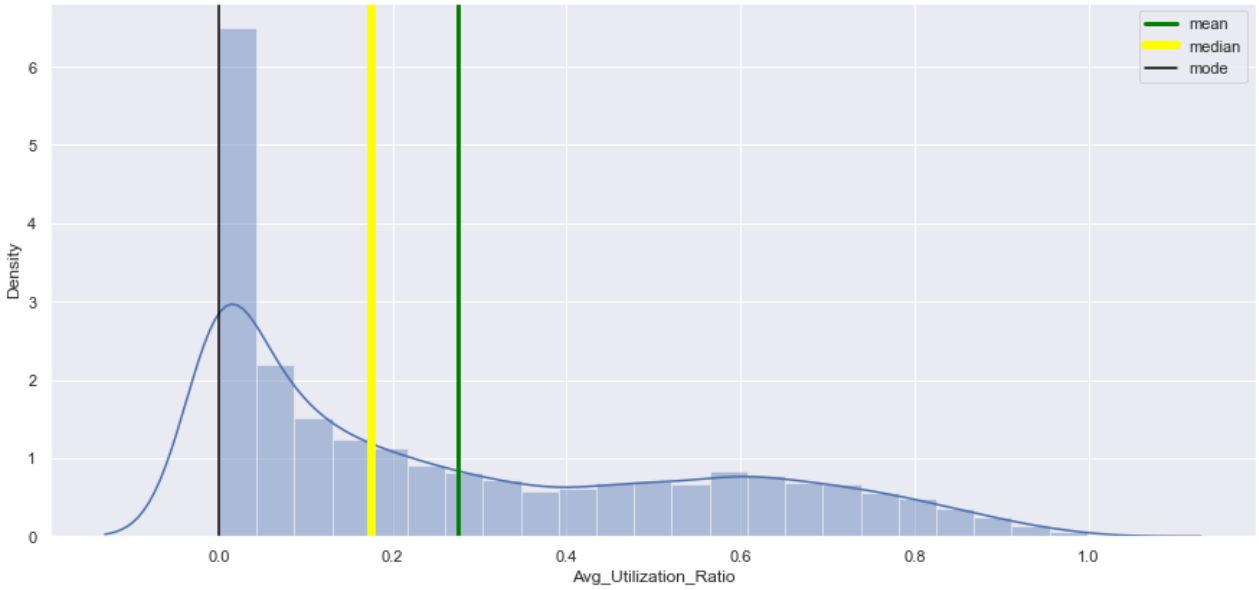
Observation on Avg_Utilization_Ratio

In [540…

```
histogram_boxplot(data.Avg_Utilization_Ratio)
```

```
Mean:0.2748935518909845
Median:0.17600000000000002
Mode:0.0
```





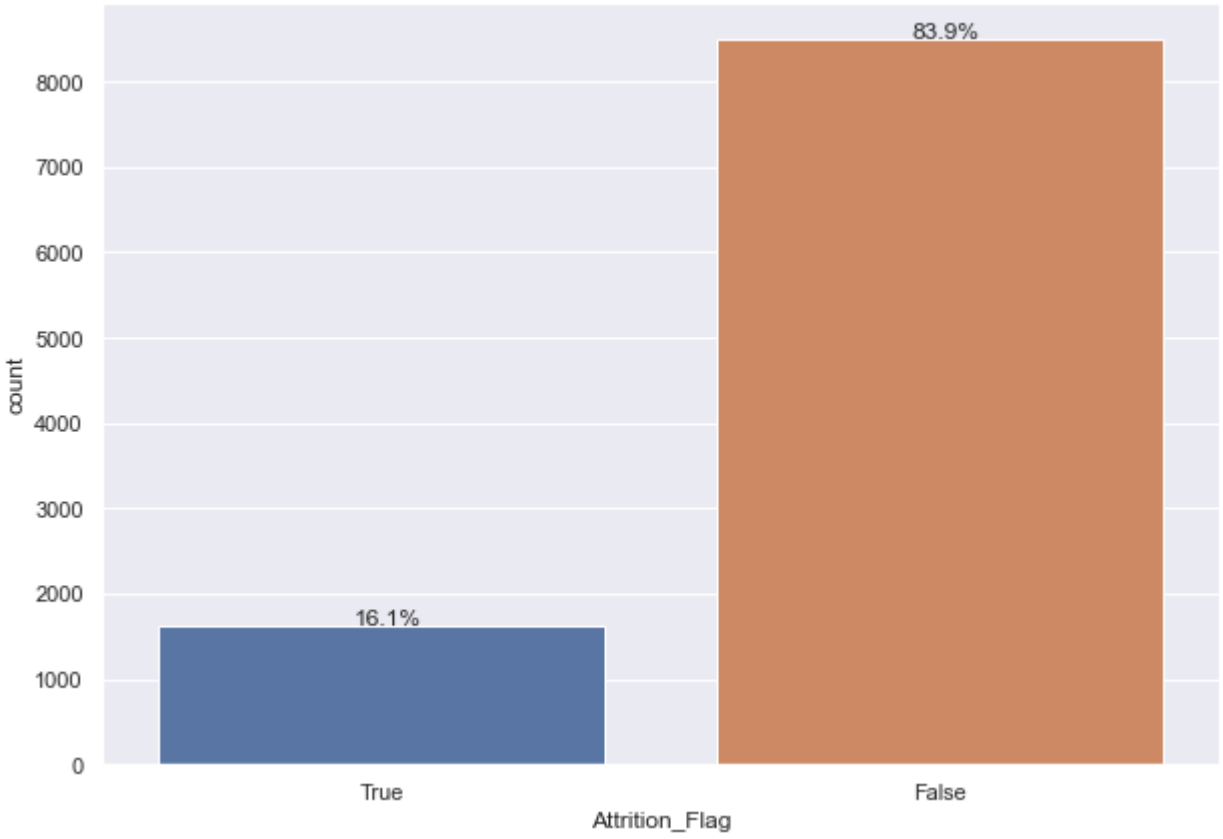
- Avg_Utilization_Ratio feature is right skewed.
- There are no outliers for the distribution.

Observations on Attrition_Flag (Dependant Variable)

In [541…

```
print('Attrition_Flag\n' , data['Attrition_Flag'].value_counts(normalize=True) , '\n')
bar_count_pct(data.Attrition_Flag)
```

```
Attrition_Flag
 False    0.83934
True     0.16066
Name: Attrition_Flag, dtype: float64

Top:False
Freq:8500
```



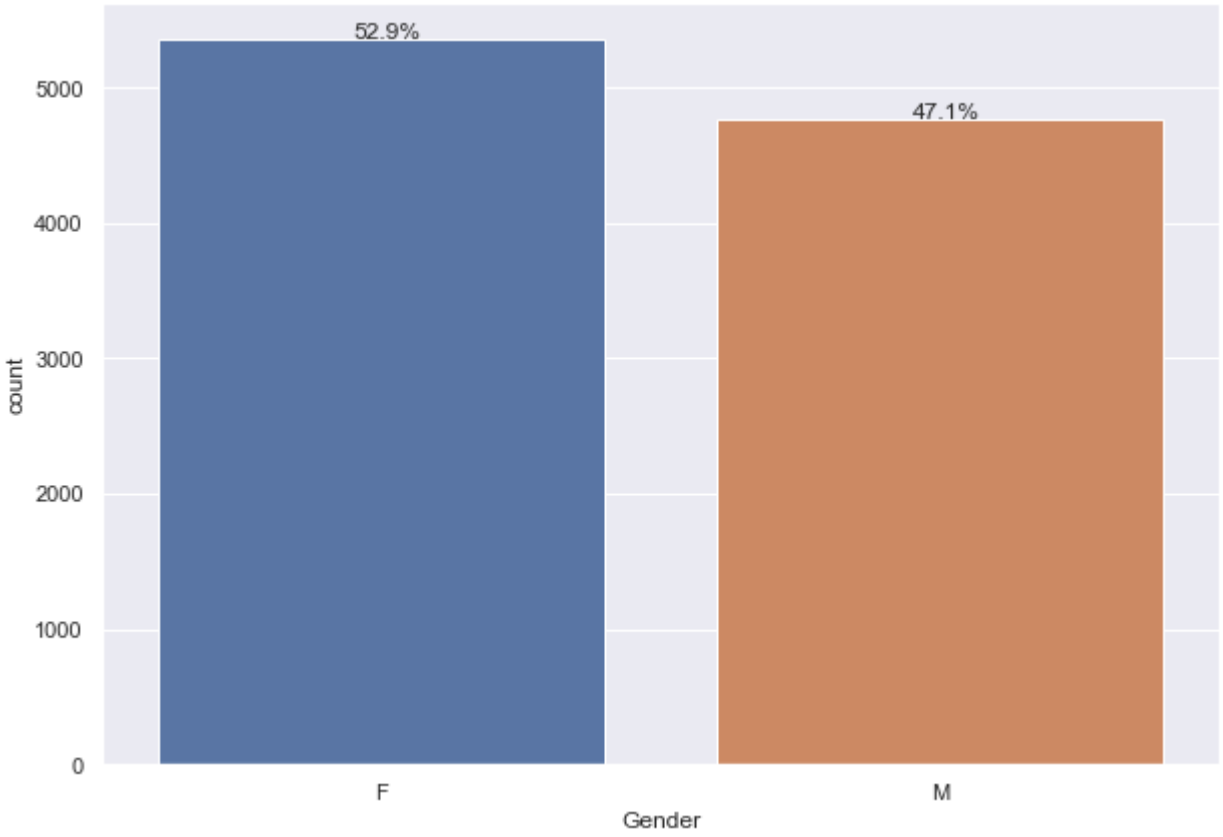
- Mode frequent ProdTaken is Existing Customer with 84%.
- Only 16% of are Attrited Customer.

Observations on Gender

In [542…

```
print('Gender\n' , data['Gender'].value_counts(normalize=True) , '\n')
bar_count_pct(data.Gender)
```

```
Gender
 F    0.529081
M    0.470919
Name: Gender, dtype: float64

Top:F
Freq:5358
```



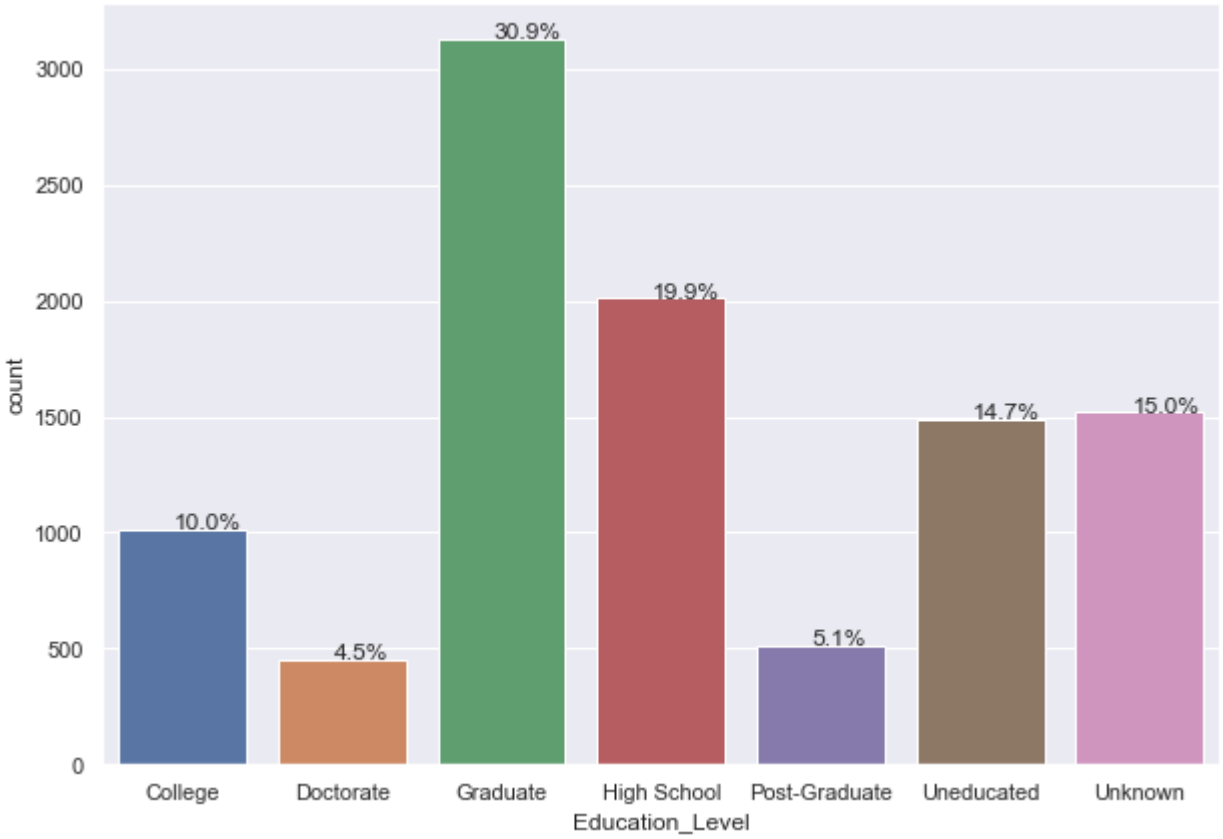
- Most common Gender is Female(F), but there is no significant difference between the genders.

Observations on Education_Level

In [543…

```
print('Education_Level\n' , data['Education_Level'].value_counts(normalize=True) , '\n'
bar_count_pct(data.Education_Level)
```

```
Education_Level
 Graduate         0.308877
High School      0.198776
Unknown          0.149995
Uneducated       0.146835
College          0.100030
Post-Graduate    0.050953
Doctorate        0.044534
Name: Education_Level, dtype: float64

Top:Graduate
Freq:3128
```



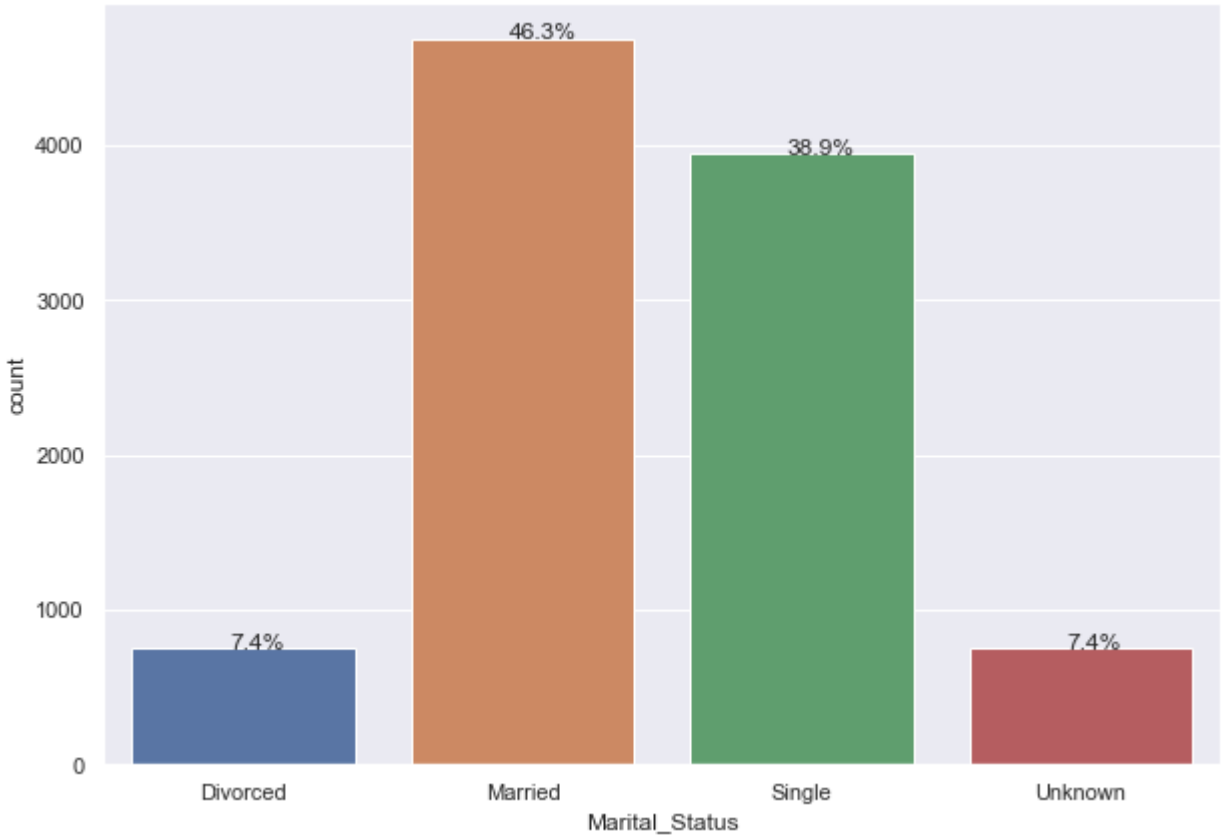
- Mode frequent Education_Level is 'Graduate' with 31%.
- Unknows Education_level has 15% of the distribution.
- There are 7 unique values.

Observations on Marital_Status

In [544…

```
print('Marital_Status\n' , data['Marital_Status'].value_counts(normalize=True) , '\n')
bar_count_pct(data.Marital_Status)
```

```
Marital_Status
 Married     0.462822
Single      0.389355
Unknown     0.073961
Divorced    0.073862
Name: Marital_Status, dtype: float64

Top:Married
Freq:4687
```



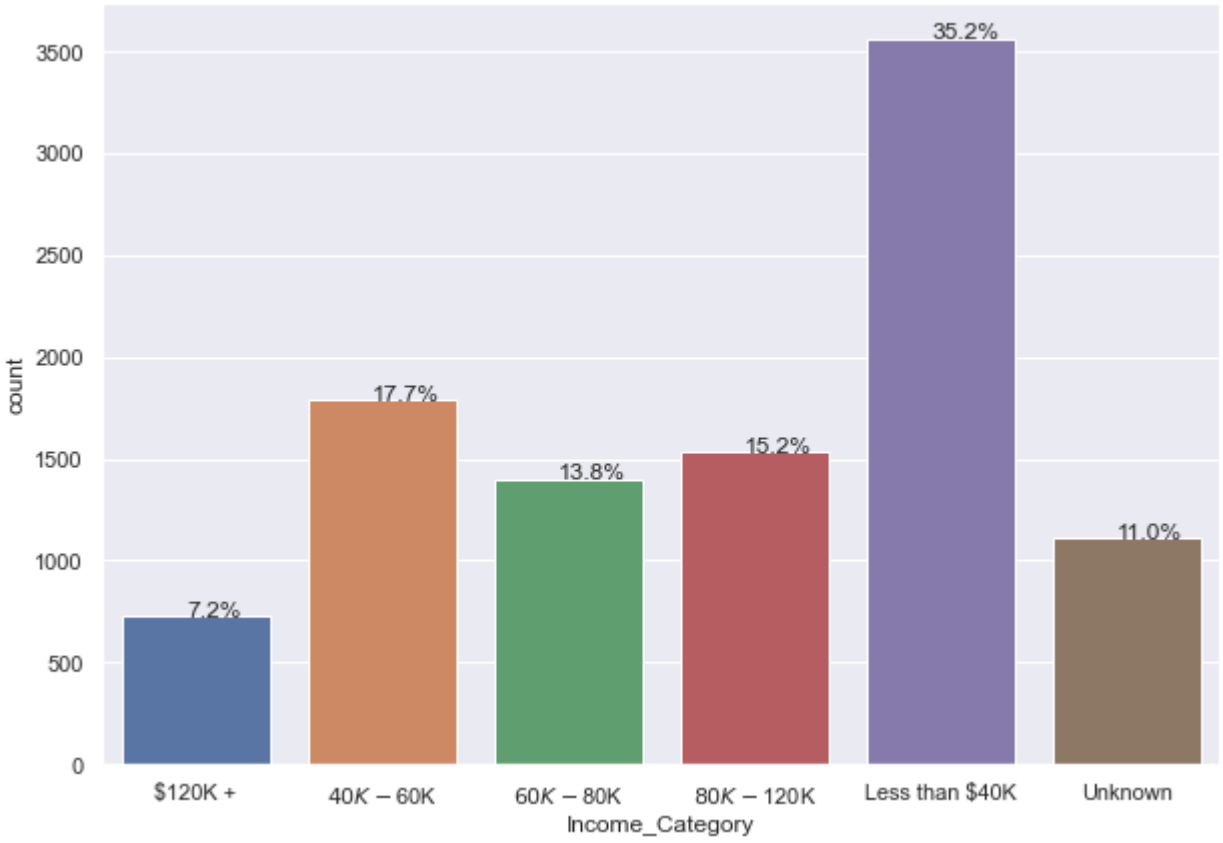
- Mode frequent Marital_Status is Married with 46%.
- Unknow Marital_Status is only 7.4% of the distribution.
- There are 4 unique values.

Observations on Income_Category

In [545…

```
print('Income_Category\n' , data['Income_Category'].value_counts(normalize=True) , '\n'
bar_count_pct(data.Income_Category)
```

```
Income_Category
 Less than $40K    0.351634
$40K - $60K       0.176755
$80K - $120K      0.151575
$60K - $80K       0.138442
Unknown           0.109805
$120K +           0.071788
Name: Income_Category, dtype: float64

Top:Less than $40K
Freq:3561
```



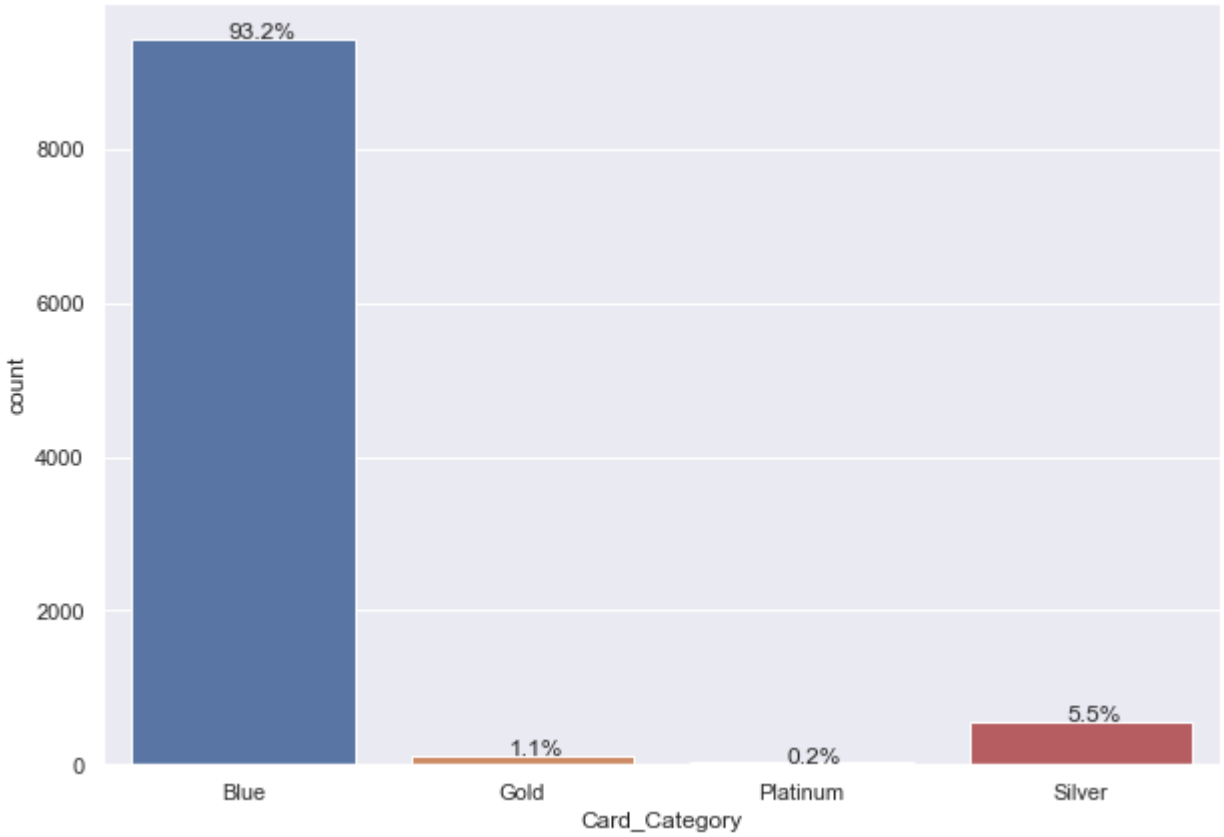
- Most frequent Income_Category is "Less than $40K" with 35%.
- Unknown Income_Category has 11% of the distribution.
- There are 6 unique values.

Observations on Card_Category

In [546…

```
print('Card_Category\n' , data['Card_Category'].value_counts(normalize=True) , '\n')
bar_count_pct(data.Card_Category)
```

```
Card_Category
 Blue        0.931767
Silver      0.054804
Gold        0.011455
Platinum    0.001975
Name: Card_Category, dtype: float64

Top:Blue
Freq:9436
```



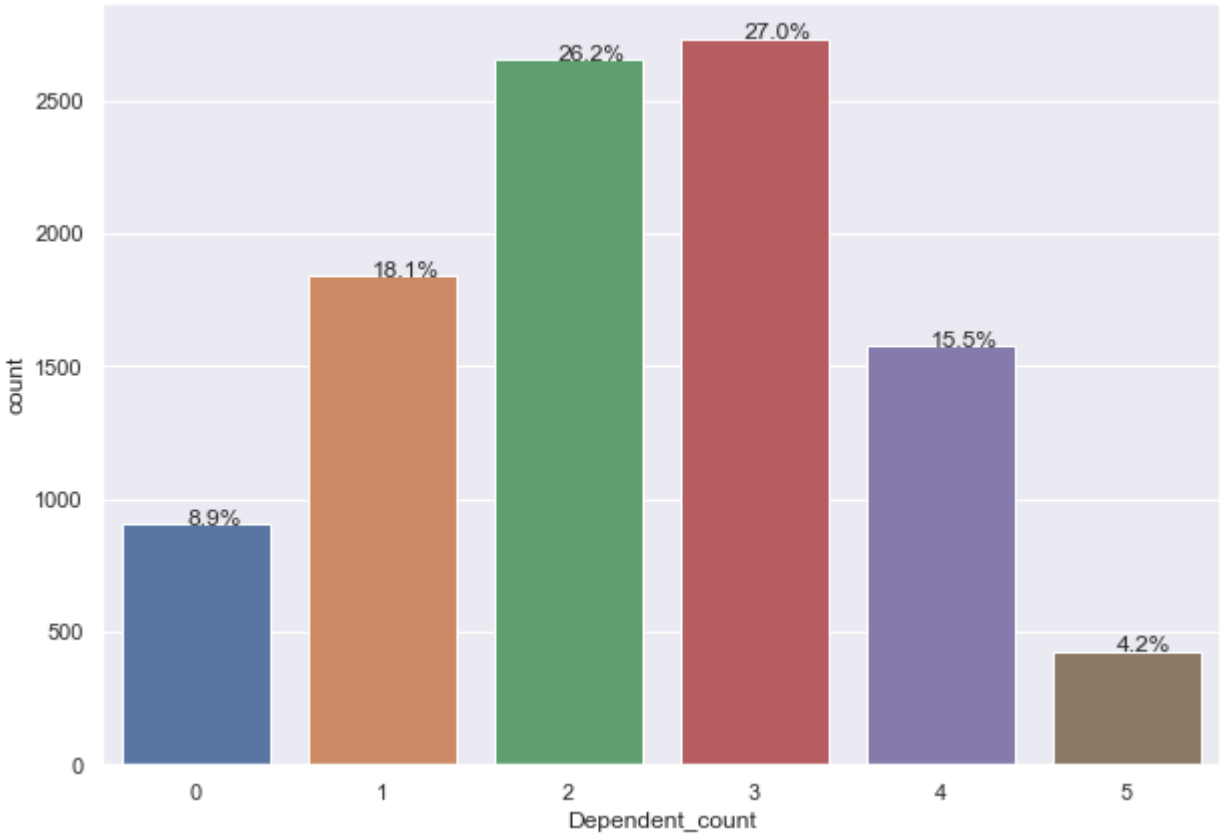
- Most frequent Card_Category is Blue with 93%.
- There are 4 unique values.

Observations on Dependent_count

In [547…

```
print('Dependent_count\n' , data['Dependent_count'].value_counts(normalize=True) , '\n'
bar_count_pct(data.Dependent_count)
```

```
Dependent_count
 3    0.269774
2    0.262170
1    0.181495
4    0.155426
0    0.089266
5    0.041868
Name: Dependent_count, dtype: float64

Top:3
Freq:2732
```



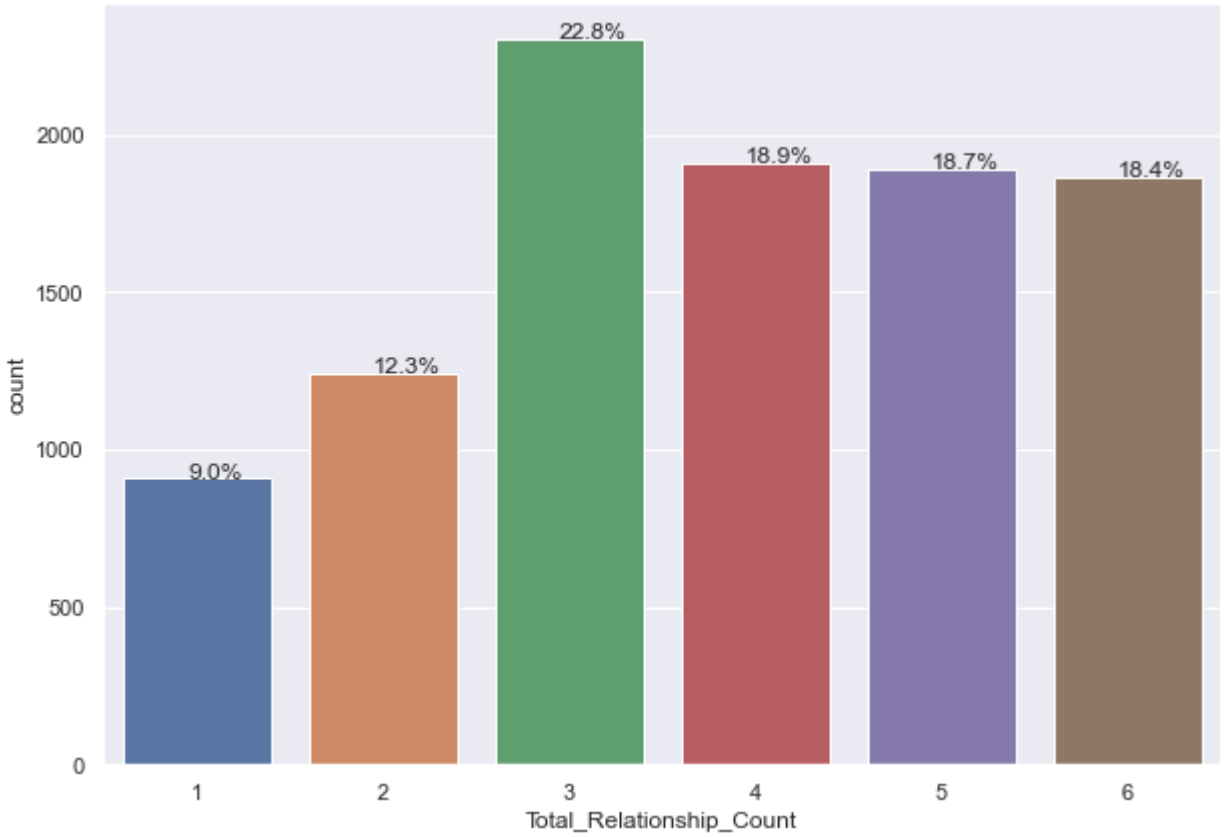
- Most frequent Dependent_count is 3 with 27%, followed by 2 with 26%.
- There are 6 unique values.

Observations on Total_Relationship_Count

In [548…

```
print('Total_Relationship_Count\n' , data['Total_Relationship_Count'].value_counts(norm
bar_count_pct(data.Total_Relationship_Count)
```

```
Total_Relationship_Count
 3    0.227609
4    0.188802
5    0.186729
6    0.184260
2    0.122741
1    0.089859
Name: Total_Relationship_Count, dtype: float64

Top:3
Freq:2305
```



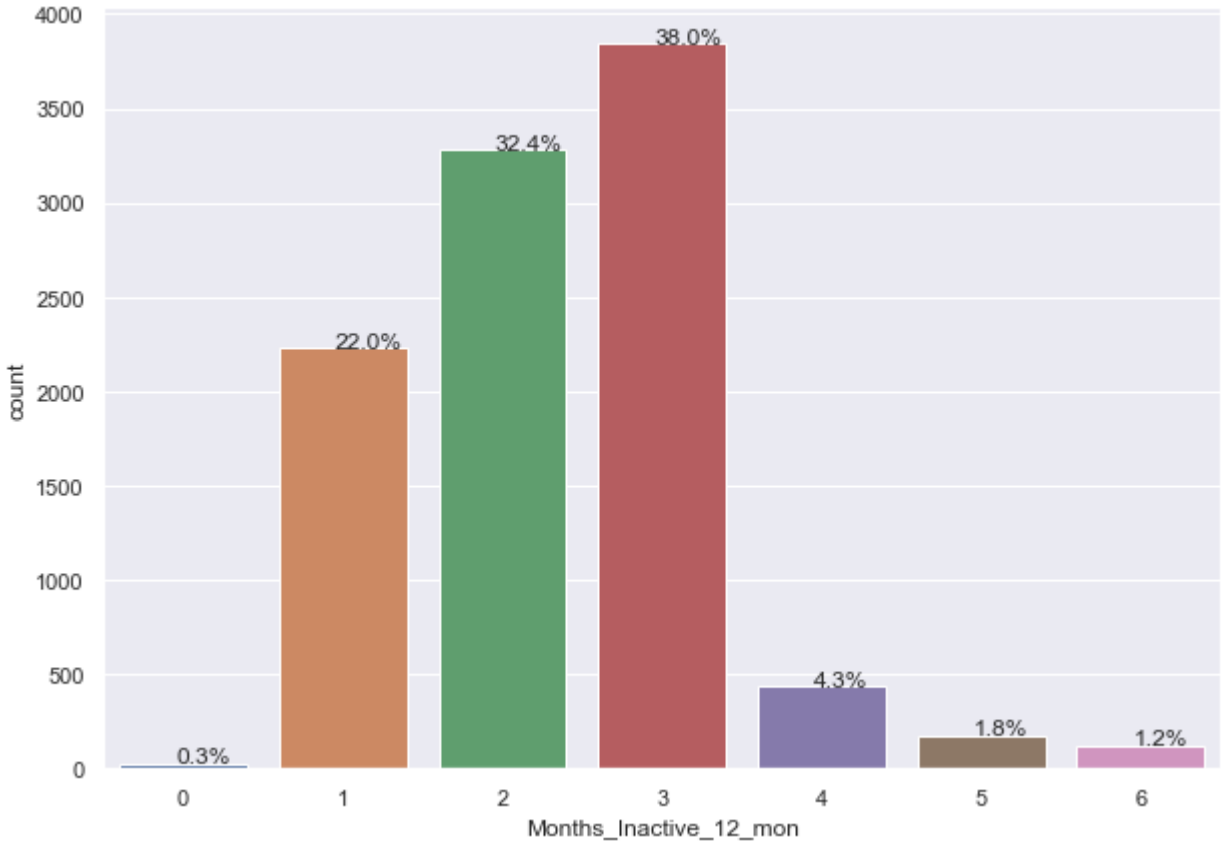
- Most frequent Total_Relationship_Count is 3 with 22.8%.
- There are 6 unique values.

Observations on Months_Inactive_12_mon

In [549…

```
print('Months_Inactive_12_mon\n' , data['Months_Inactive_12_mon'].value_counts(normaliz
bar_count_pct(data.Months_Inactive_12_mon)
```

```
Months_Inactive_12_mon
 3    0.379777
2    0.324084
1    0.220500
4    0.042954
5    0.017577
6    0.012244
0    0.002864
Name: Months_Inactive_12_mon, dtype: float64

Top:3
Freq:3846
```



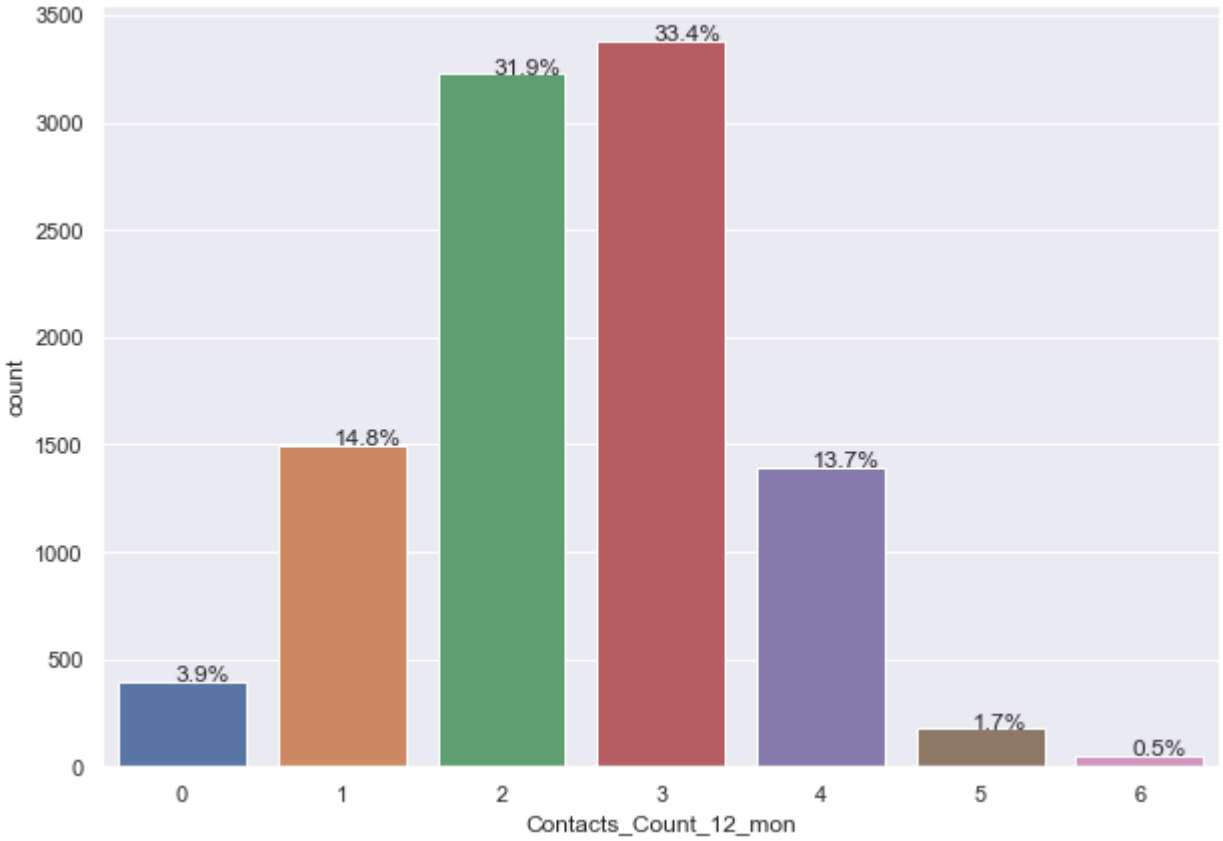
- Most frequent Months_Inactive_12_mon is 3 with 38%.
- There are 7 unique values.

Observations on Contacts_Count_12_mon

In [550…

```
print('Contacts_Count_12_mon\n' , data['Contacts_Count_12_mon'].value_counts(normalize=
bar_count_pct(data.Contacts_Count_12_mon)
```

```
Contacts_Count_12_mon
 3    0.333761
2    0.318653
1    0.148020
4    0.137454
0    0.039400
5    0.017379
6    0.005332
Name: Contacts_Count_12_mon, dtype: float64

Top:3
Freq:3380
```



- Most frequent Contacts_Count_12_mon is 3 with 33.4%.
- There are 7 unique values.

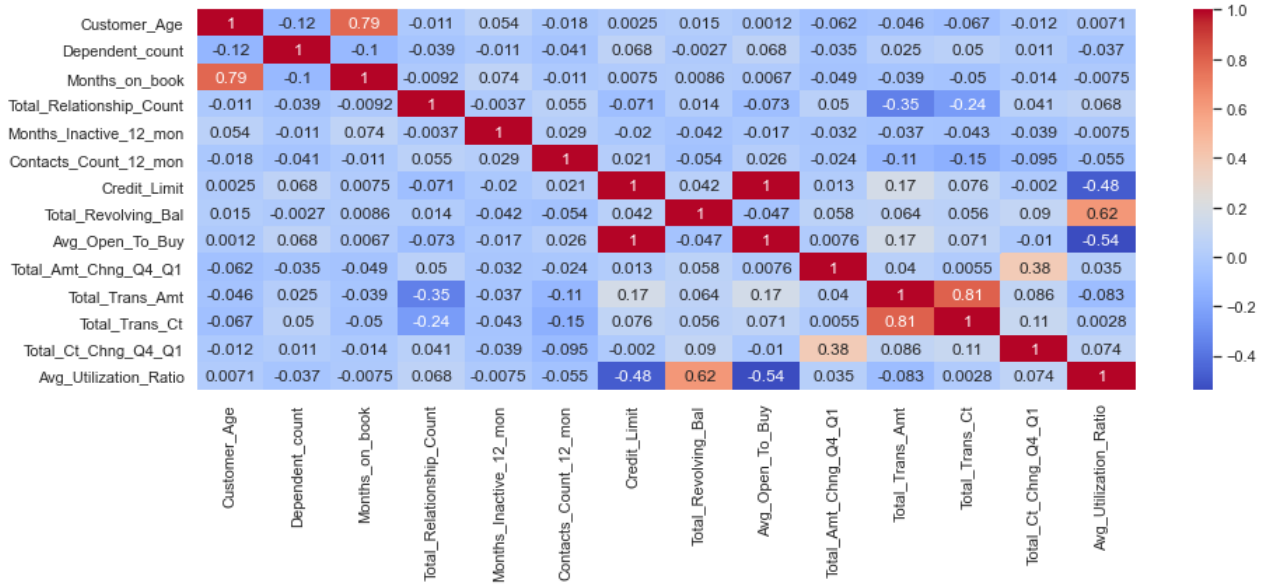
In [ ]:

Bivariate Analysis

In [ ]:

In [551…

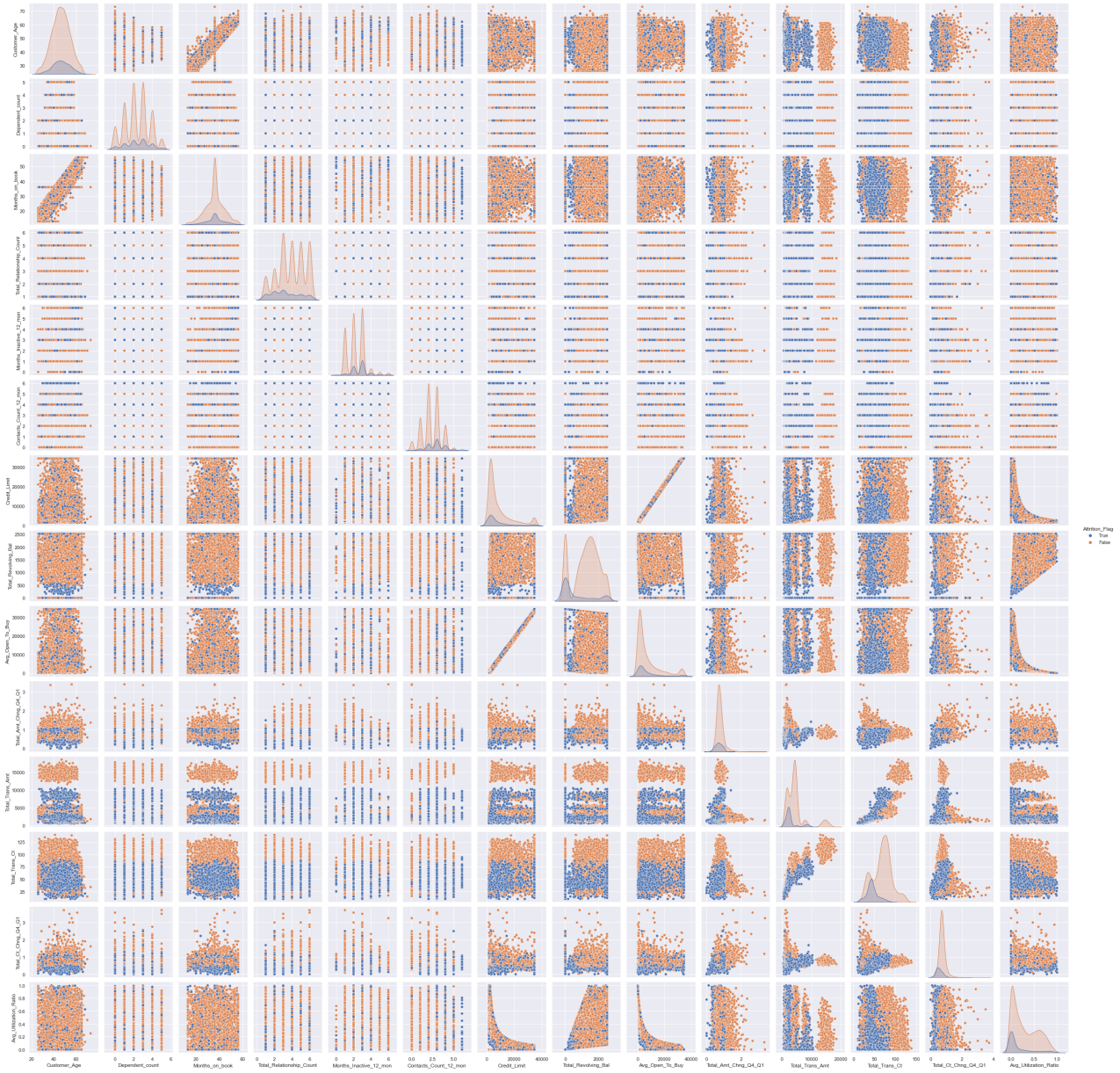
```
plt.figure(figsize=(15,5))
sns.heatmap(data.corr(),annot=True, cmap="coolwarm")
plt.show()
```



- The pairs that have a positive/high (>0.5) correlation are:
  - Credit_limit/Avg_Open_To_Buy
  - Total_Trans_Ct/Total_Trans_Amt
  - Customer_Age/Months_on_book
  - Total_Revolving_Bal/Avg_Utilization_Ratio
- The pairs that have a negative/high (<-0.5) correlation are:
  - Avg_Utilization_Ratio/Avg_Open_To_Buy

In [552…
```
sns.pairplot(data, hue="Attrition_Flag")
```

Out[552…
```
<seaborn.axisgrid.PairGrid at 0x1b7de5fa670>
```



Bivariate analysis every possible attribute pair in relation to Attrition_Flag

In [553…

```
### Function to plot distributions and Boxplots of customers
def dist_catplot(x,target='Attrition_Flag'):
    fig,axs = plt.subplots(2,2,figsize=(12,10))
    axs[0, 0].set_title('Distribution of a customer with Attrition_Flag')
    sns.distplot(data[(data[target] == 1)][x],ax=axs[0,0],color='teal')
    axs[0, 1].set_title("Distribution of a customer with out Attrition_Flag")
    sns.distplot(data[(data[target] == 0)][x],ax=axs[0,1],color='orange')
    axs[1,0].set_title('Boxplot with respect to Attrition_Flag')
    sns.boxplot(data[target],data[x],ax=axs[1,0],palette='gist_rainbow')
    axs[1,1].set_title('Boxplot with respect to Attrition_Flag - Without outliers')
    sns.boxplot(data[target],data[x],ax=axs[1,1],showfliers=False,palette='gist_rainbow
    plt.tight_layout()
    plt.show()
```

In [554…

```
### Function to plot stacked bar charts for categorical columns
def stacked_plot(x):
    sns.set()
```

```
    ## crosstab
    tab1 = pd.crosstab(x,data['Attrition_Flag'],margins=True).sort_values(by=1,ascendin
    print(tab1)
    print('-'*120)
    ## visualising the cross tab
    tab = pd.crosstab(x,data['Attrition_Flag'],normalize='index').sort_values(by=1,asce
    tab.plot(kind='bar',stacked=True,figsize=(17,7))
    plt.legend(loc='lower left', frameon=False,)
    plt.legend(loc="upper left", bbox_to_anchor=(1,1))
    plt.show()
```

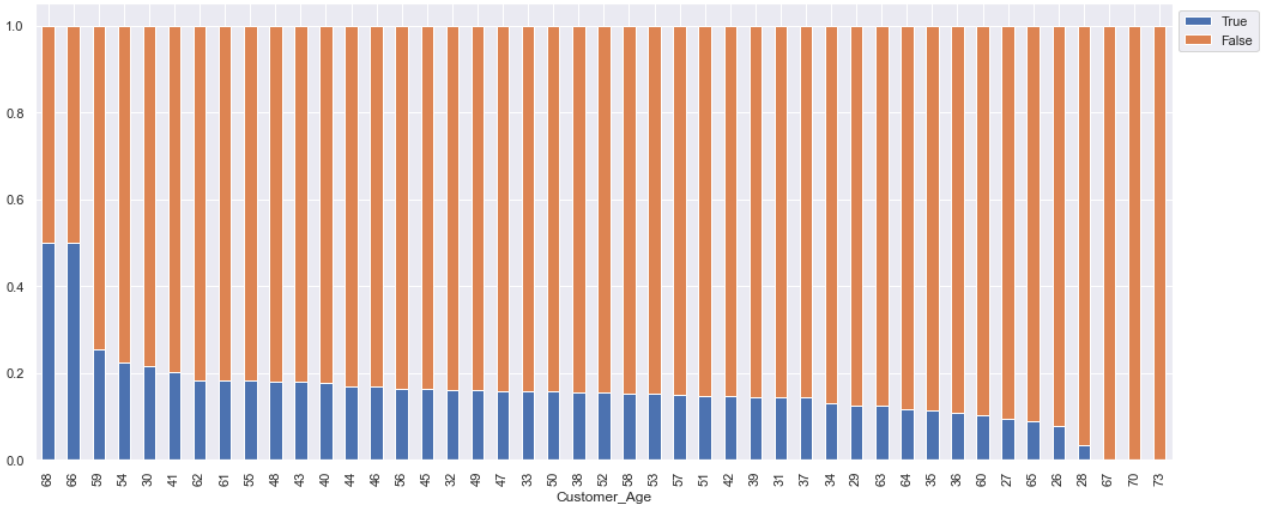
Customer_Age vs Attrition_Flag

In [555…

```
stacked_plot(data['Customer_Age'])
```

Attrition_Flag	True	False	All
Customer_Age			
All	1627	8500	10127
43	85	388	473
48	85	387	472
44	84	416	500
46	82	408	490
45	79	407	486
49	79	416	495
47	76	403	479
41	76	303	379
50	71	381	452
54	69	238	307
40	64	297	361
42	62	364	426
53	59	328	387
52	58	318	376
51	58	340	398
55	51	228	279
39	48	285	333
38	47	256	303
56	43	219	262
59	40	117	157
37	37	223	260
57	33	190	223
58	24	133	157
36	24	197	221
35	21	163	184
33	20	107	127
34	19	127	146
32	17	89	106
61	17	76	93
62	17	76	93
30	15	55	70
31	13	78	91
60	13	114	127
65	9	92	101
63	8	57	65
29	7	49	56
26	6	72	78
64	5	38	43
27	3	29	32
28	1	28	29
66	1	1	2
68	1	1	2
67	0	4	4
70	0	1	1
73	0	1	1

```
--------------------------------------------------------------------------------
--------------------------------
```



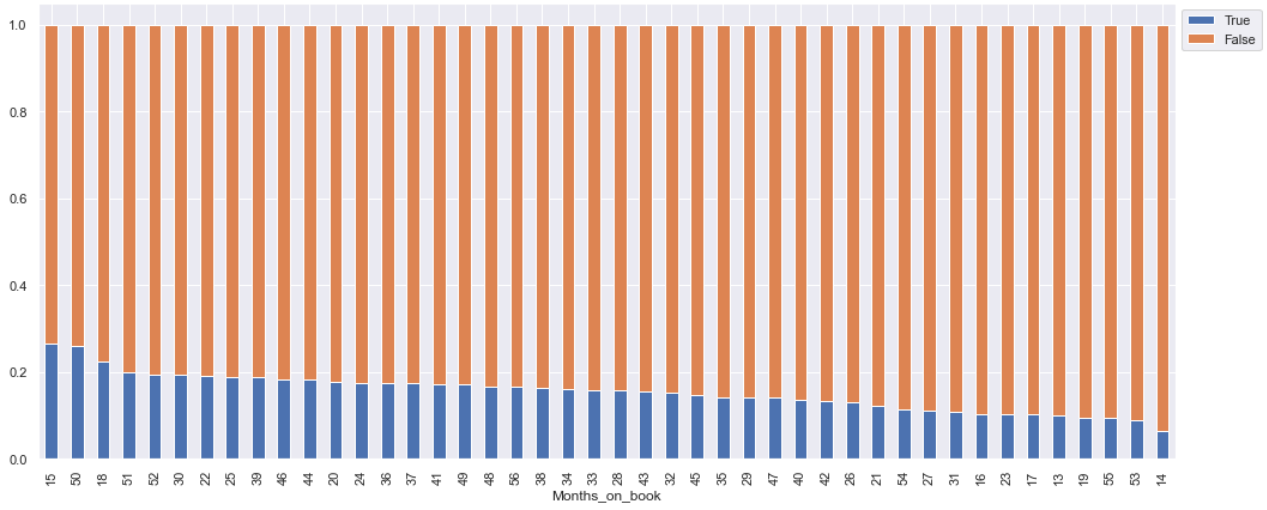
- Propertianlally, There is significat difference in Ages 66-68 for customer that have Attrition_Flag = true.

Months_on_book vs Attrition_Flag

In [556…
```
stacked_plot(data['Months_on_book'])
```

Attrition_Flag	True	False	All
Months_on_book			
All	1627	8500	10127
36	430	2033	2463
39	64	277	341
37	62	296	358
30	58	242	300
38	57	290	347
34	57	296	353
41	51	246	297
33	48	257	305
40	45	288	333
35	45	272	317
32	44	245	289
28	43	232	275
44	42	188	230
43	42	231	273
46	36	161	197
42	36	235	271
29	34	207	241
31	34	284	318
45	33	194	227
25	31	134	165
24	28	132	160
48	27	135	162
50	25	71	96
49	24	117	141
26	24	162	186
47	24	147	171
27	23	183	206
22	20	85	105
56	17	86	103
51	16	64	80
18	13	45	58
20	13	61	74

```
52                 12     50     62
23                 12    104    116
21                 10     73     83
15                  9     25     34
53                  7     71     78
13                  7     63     70
19                  6     57     63
54                  6     47     53
17                  4     35     39
55                  4     38     42
16                  3     26     29
14                  1     15     16
--------------------------------------------------------------------------------
--------------------------------
```

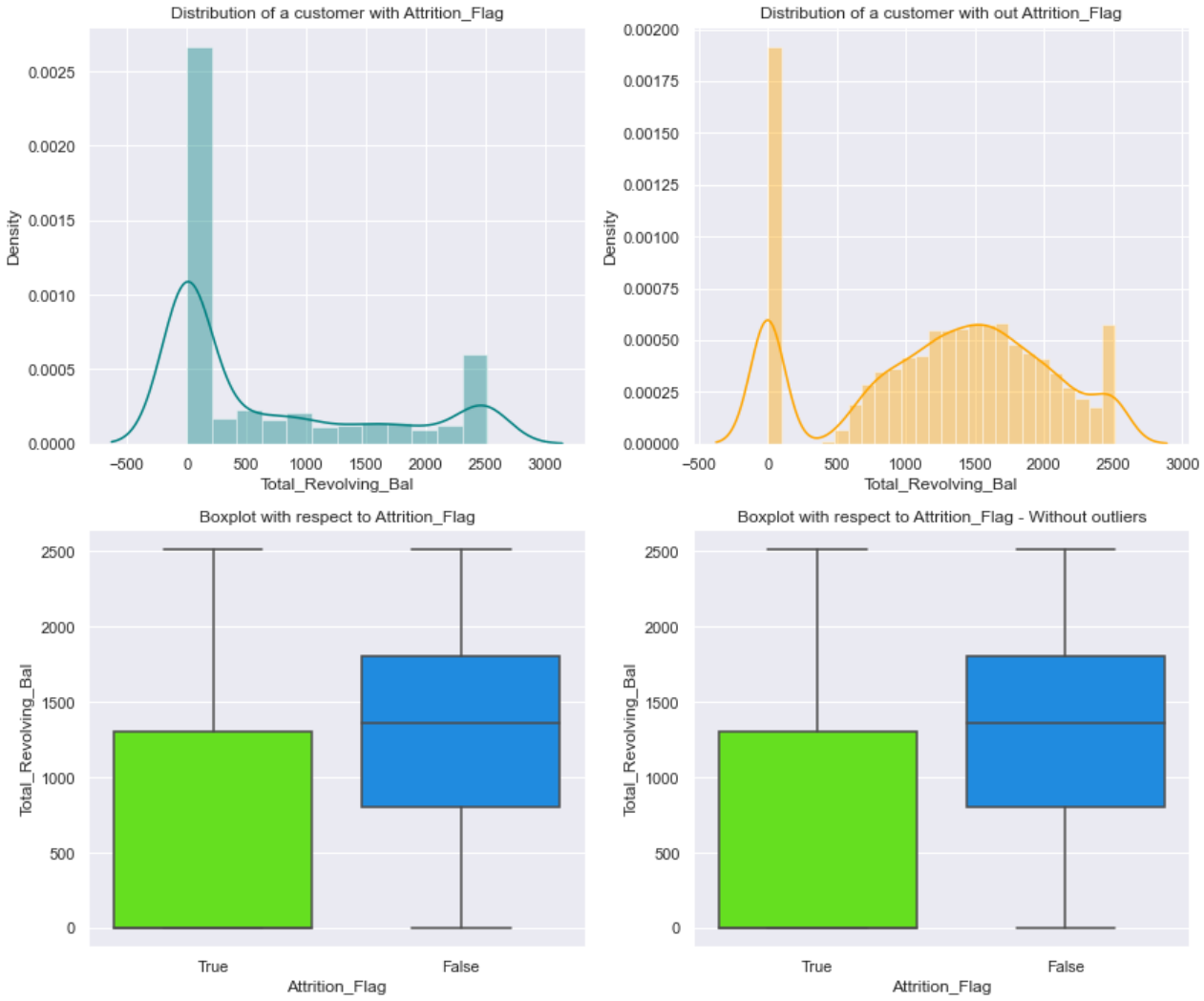


- Proportionaly, there is no significat difference in `Months_on_book` for customer that have Attrition_Flag and those who do not.

Total_Revolving_Bal vs Attrition_Flag

In [557…

```
dist_catplot('Total_Revolving_Bal')
```

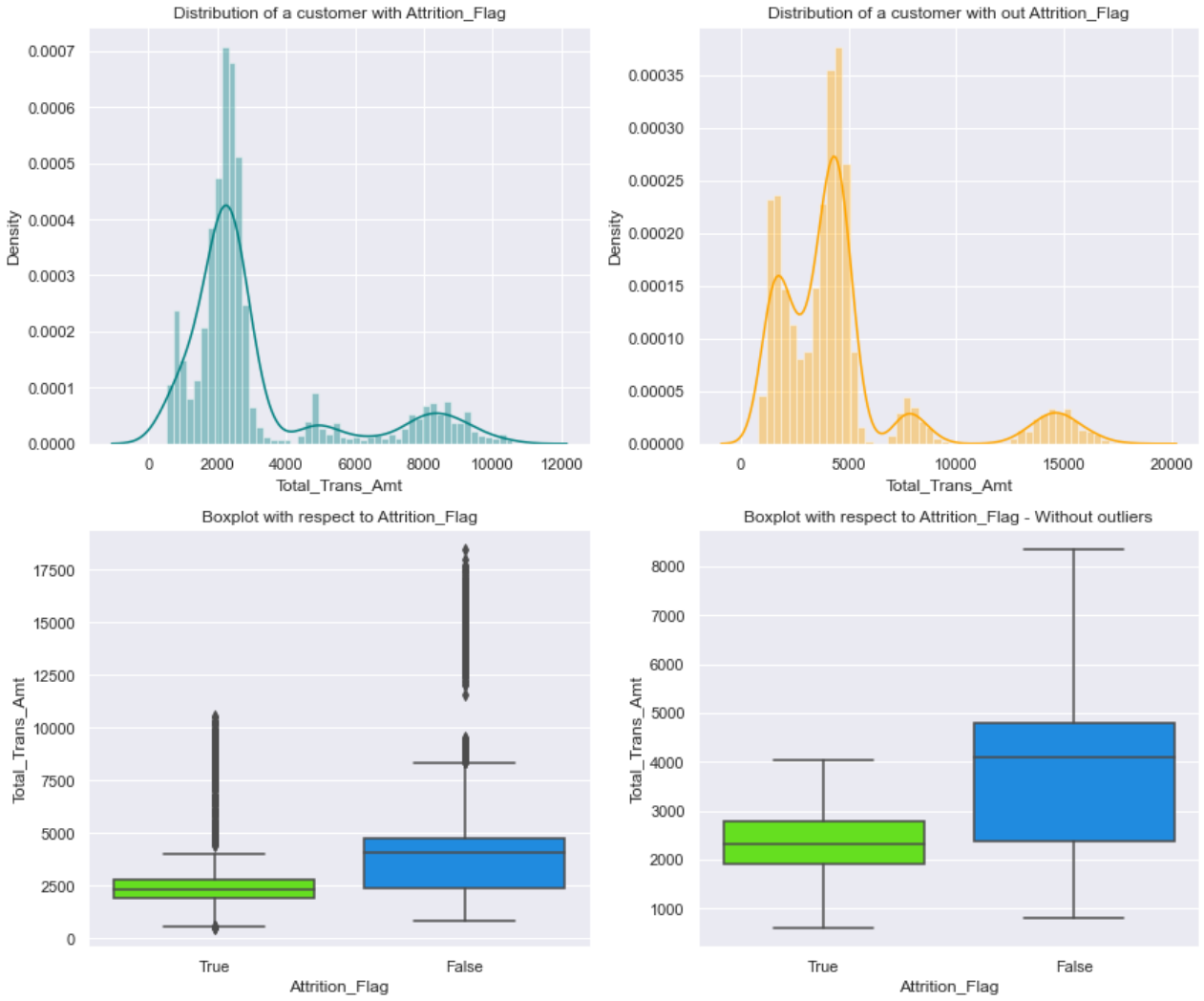


- There is no major diference in `Total_Revolving_Bal` with regard to Attrition_Flag.

Total_Trans_Amt vs Attrition_Flag

In [558…

```
dist_catplot('Total_Trans_Amt')
```

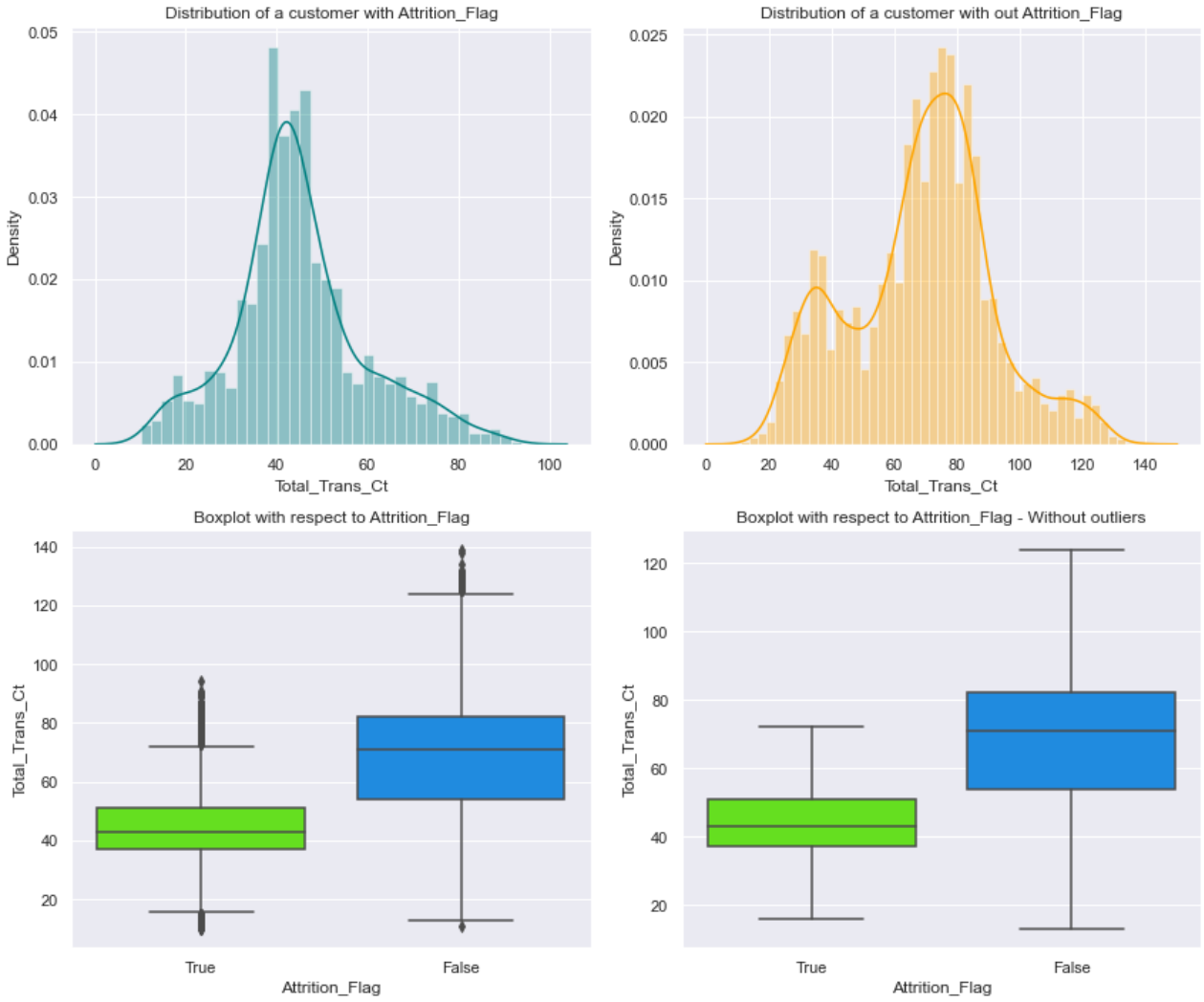


- With or without outliers in the comparison, there does not seem to be major difference in `Total_Trans_Amt` in regards to Attrition_Flag.

Total_Trans_Ct vs Attrition_Flag

In [559…

```
dist_catplot('Total_Trans_Ct')
```

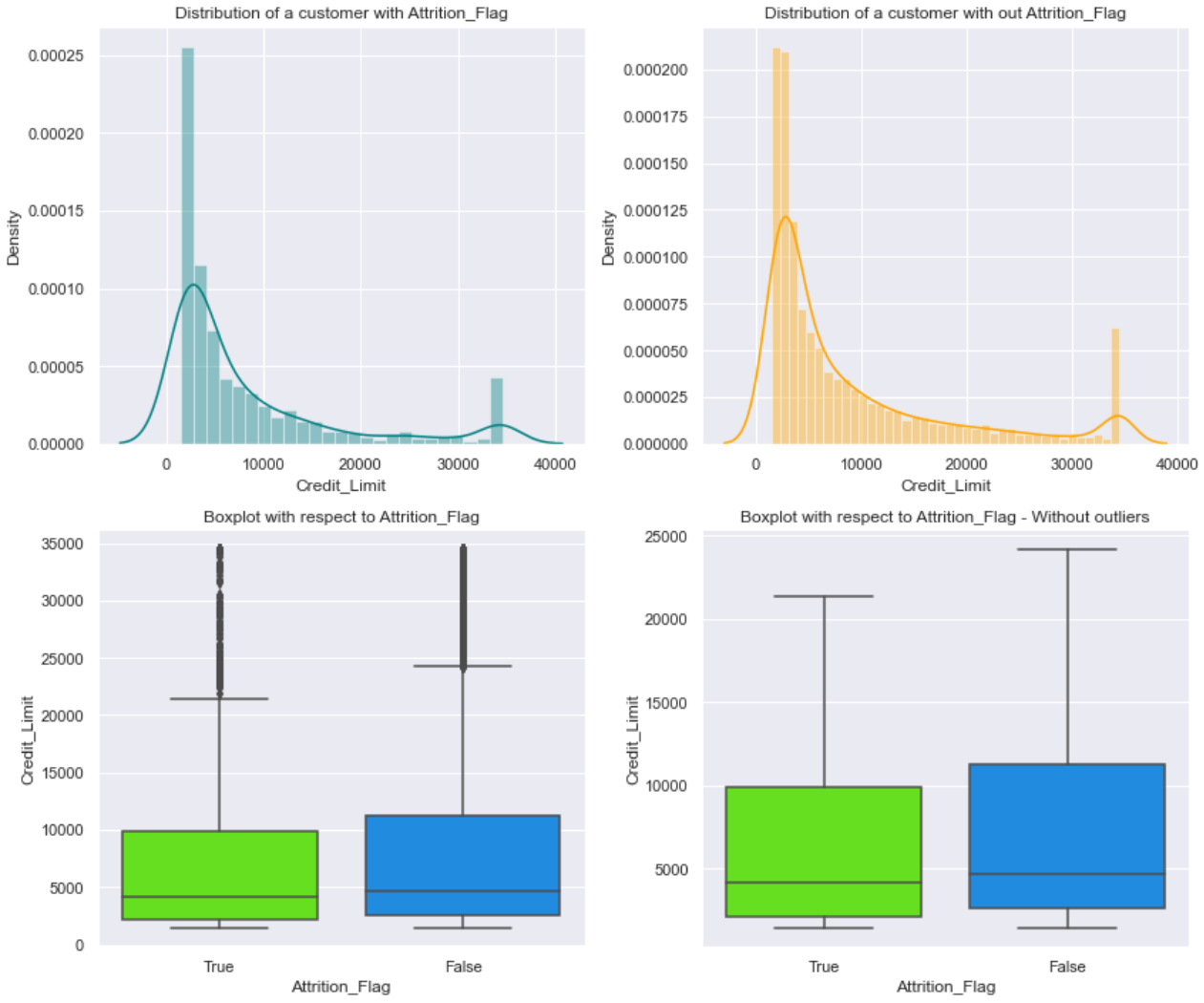


- With or without outliers in the comparison, there does not seem to be major difference in `Total_Trans_Ct` in regards to Attrition_Flag.

Credit_Limit vs Attrition_Flag

In [560…

```
dist_catplot('Credit_Limit')
```

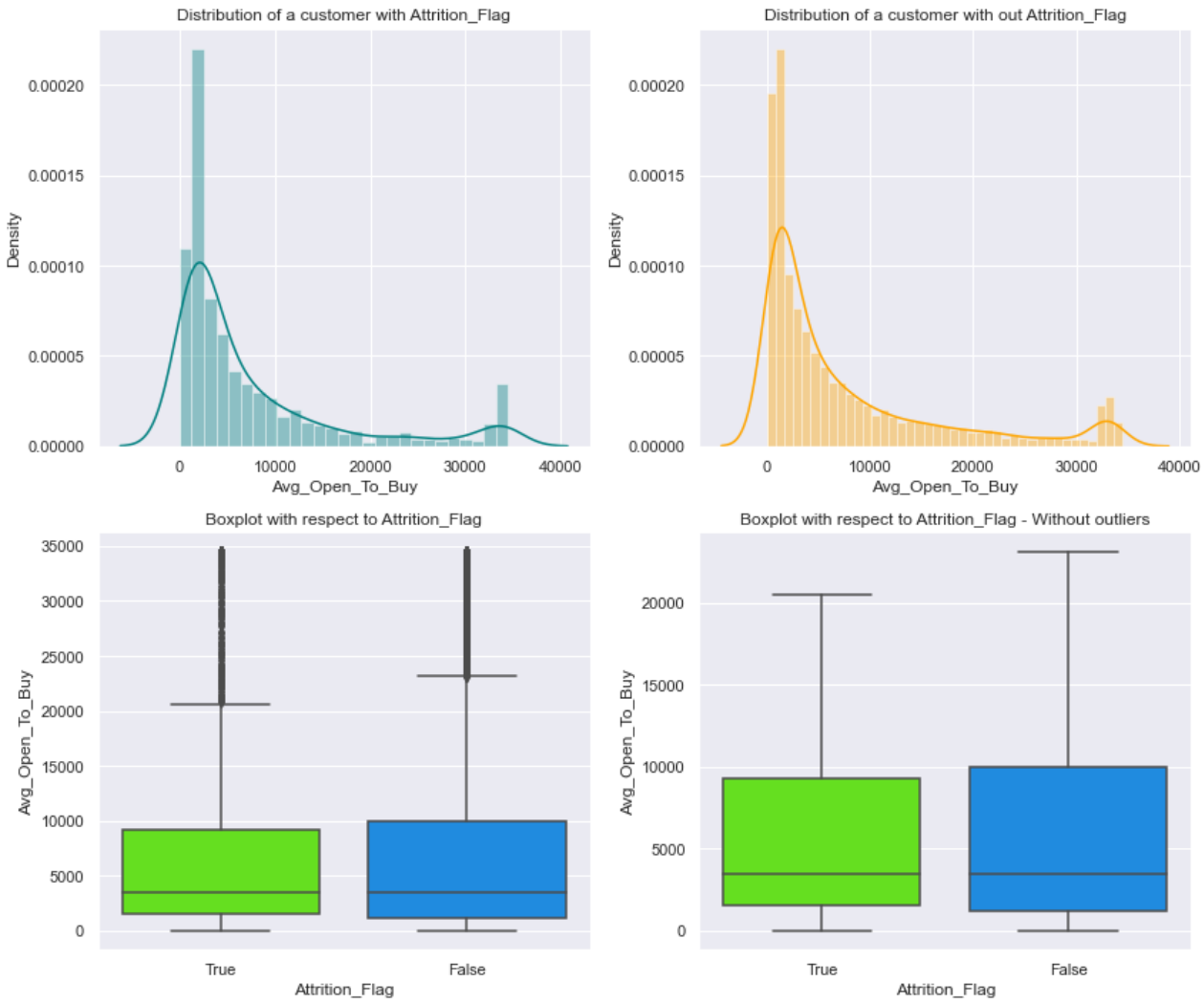


- With or without outliers in the comparison, there does not seem to be major difference in `Credit_Limit` in regards to Attrition_Flag.

Avg_Open_To_Buy vs Attrition_Flag

In [561…

```
dist_catplot('Avg_Open_To_Buy')
```

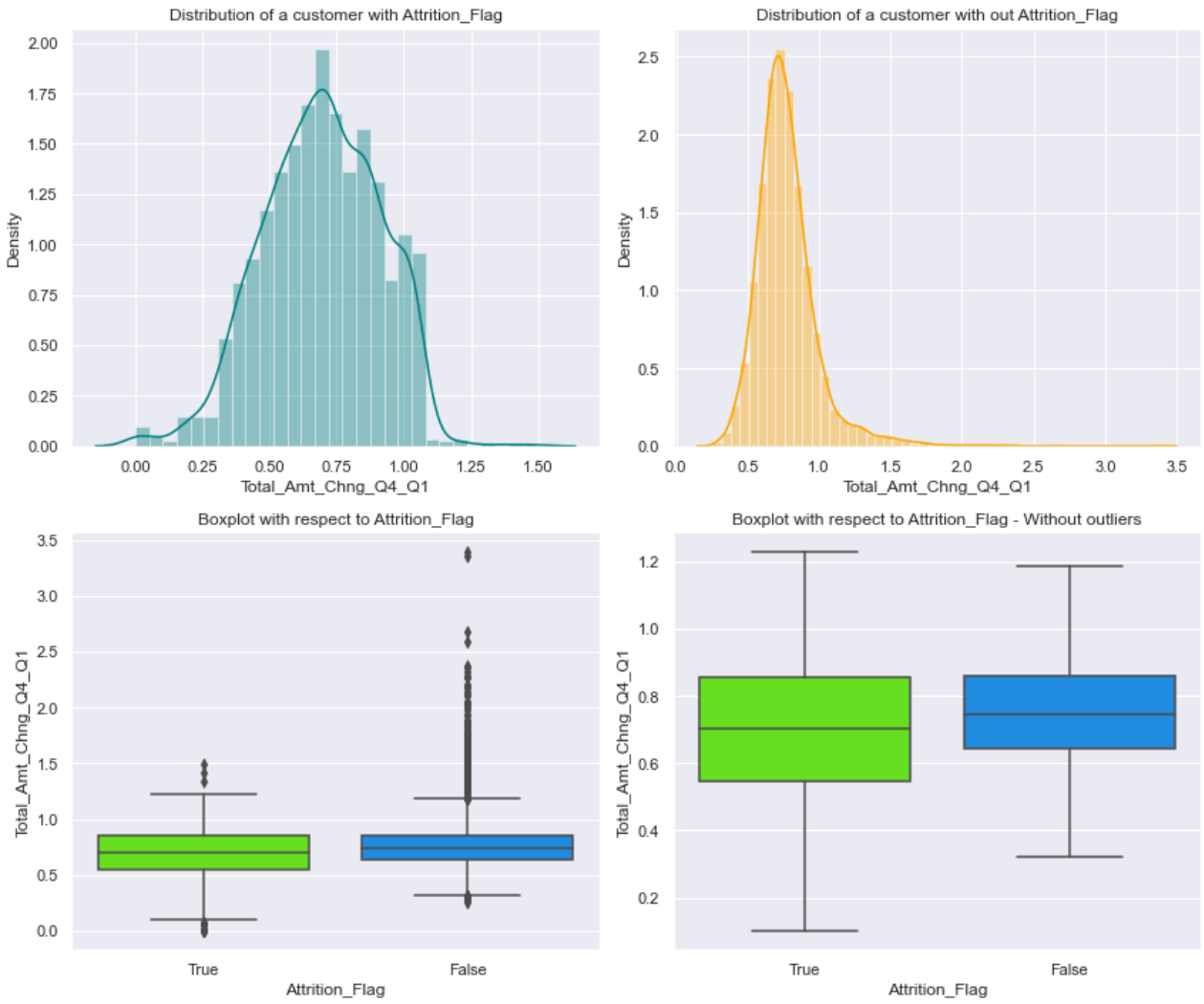


- There is no significant difference on `Avg_Open_To_Buy` when compared with or without outliers. In regards to Attrition_Flag.

Total_Amt_Chng_Q4_Q1 vs Attrition_Flag

In [562…

```
dist_catplot('Total_Amt_Chng_Q4_Q1')
```

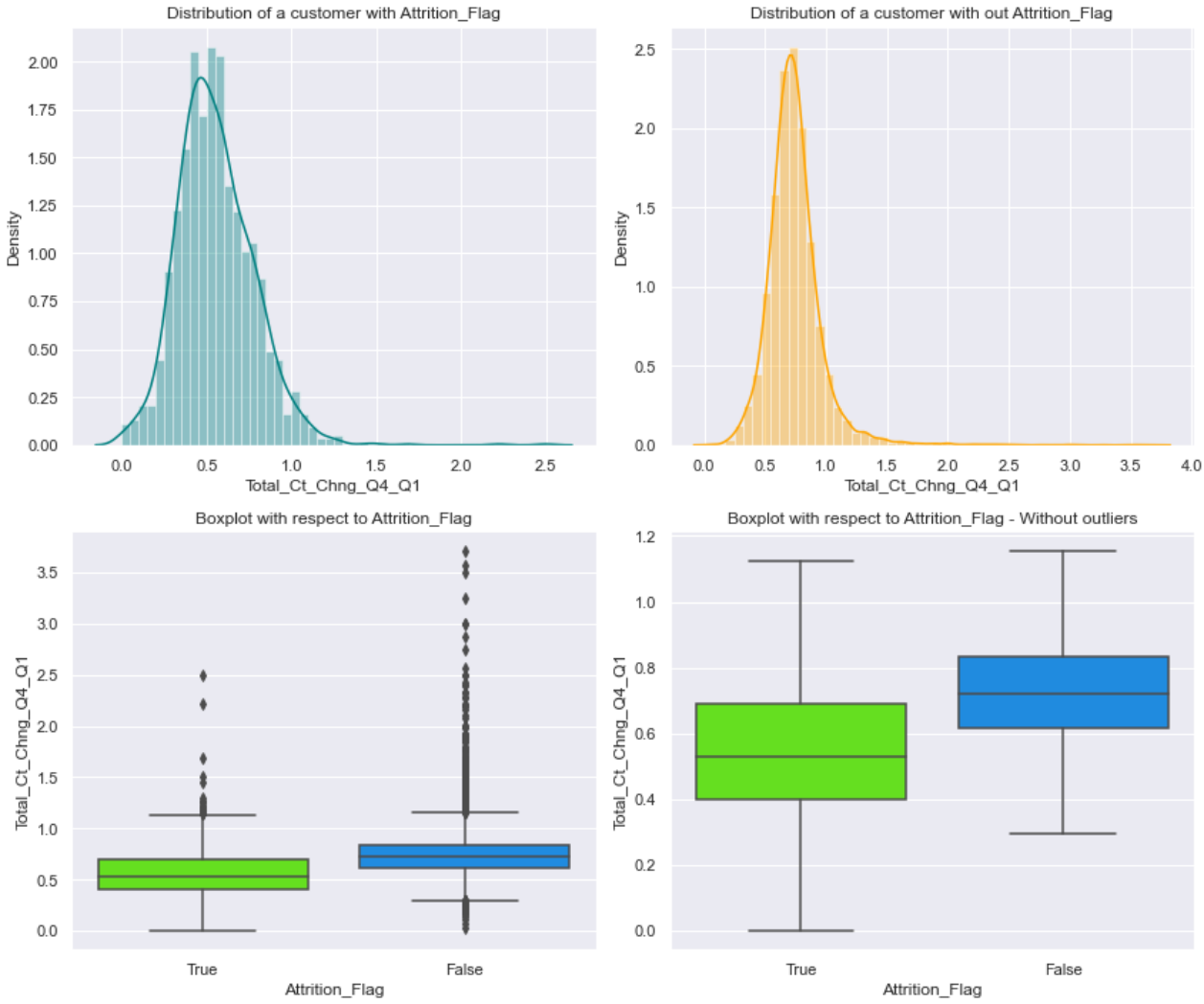


- Total_Amt_Chng_Q4_Q1 without Attrition_Flag(0) seem to be rightly skewed, compared to Attrition_Flag(1) which is more normally distibuted.
- It looks like the outliers are playuing a bigger role in skewing the data towards Total_Amt_Chng_Q4_Q1 when Attrition_Flag is false.

Total_Ct_Chng_Q4_Q1 vs Attrition_Flag

In [563…

```
dist_catplot('Total_Ct_Chng_Q4_Q1')
```

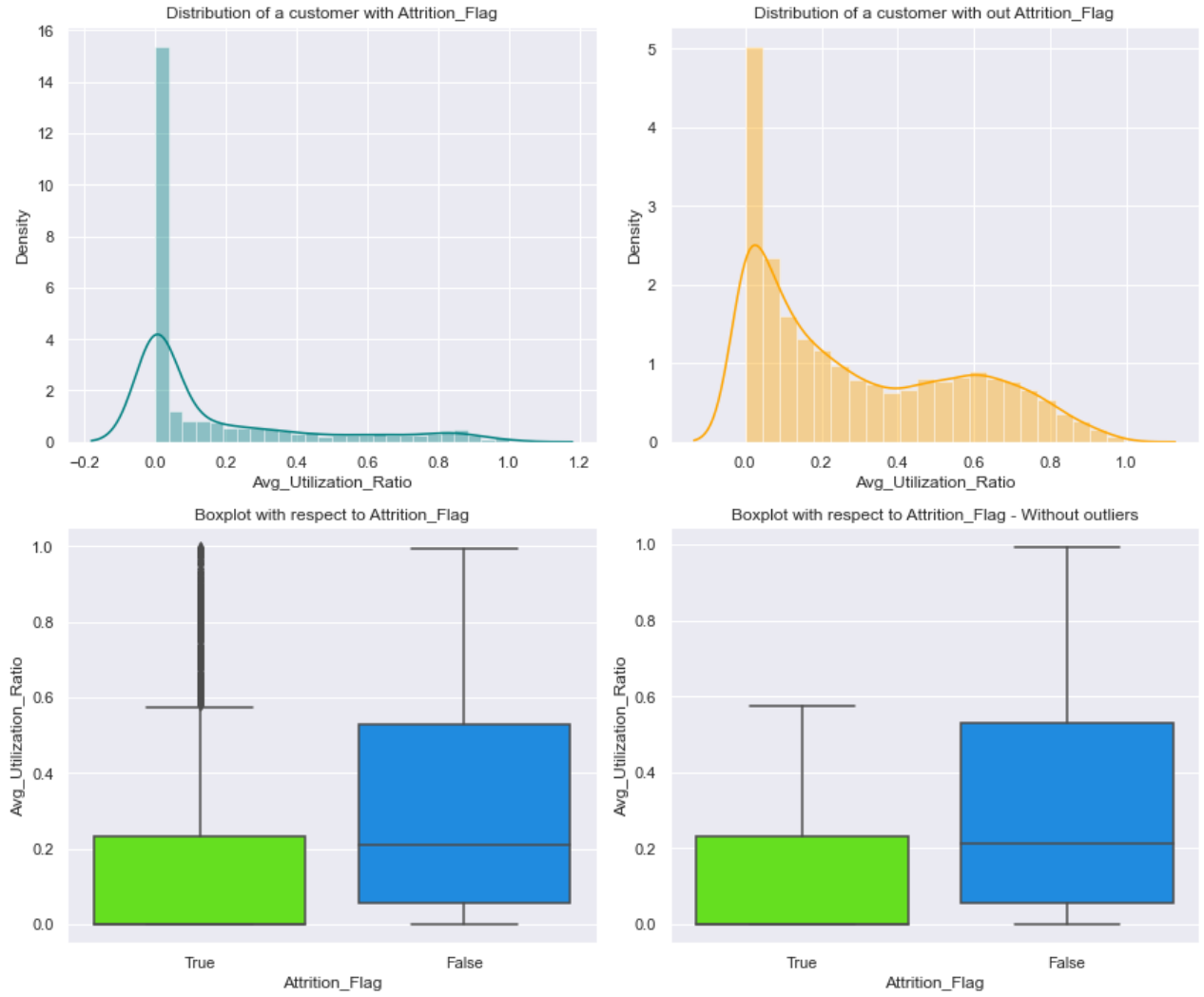


- There is no significant difference on `Total_Ct_Chng_Q4_Q1` when compared with or without outliers. In regards to Attrition_Flag.

Avg_Utilization_Ratio vs Attrition_Flag

In [564…

```
dist_catplot('Avg_Utilization_Ratio')
```



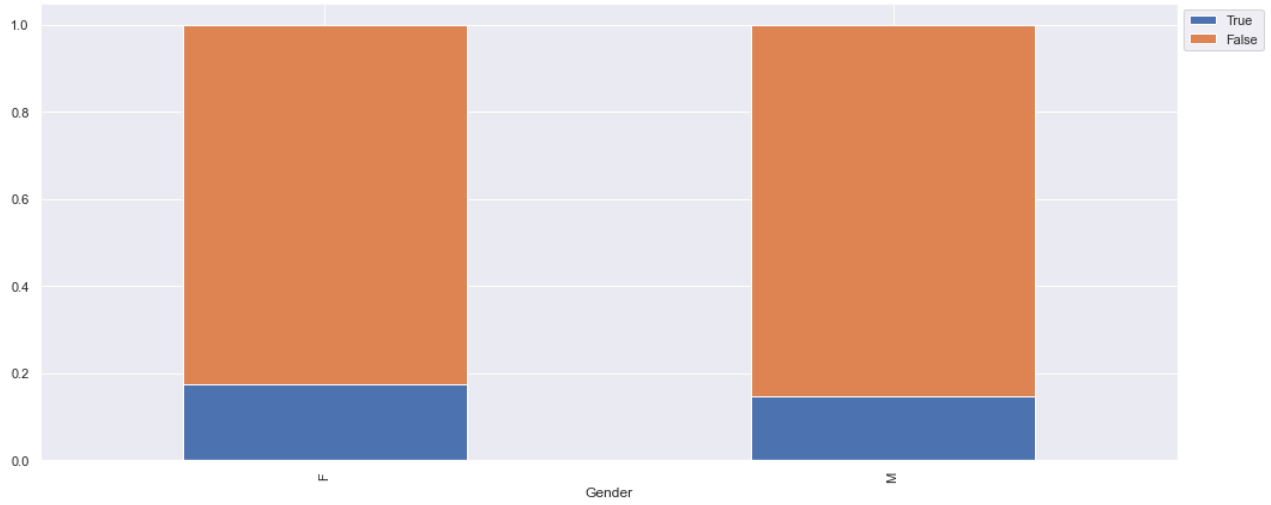
- There is no significant difference on `Avg_Utilization_Ratio` when compared with or without outliers. In regards to Attrition_Flag.

Gender vs Attrition_Flag

In [565…

```
stacked_plot(data['Gender'])
```

```
Attrition_Flag  True  False    All
Gender
All             1627   8500  10127
F                930   4428   5358
M                697   4072   4769
------------------------------------------------------------------------------------------------
--------------------------------
```

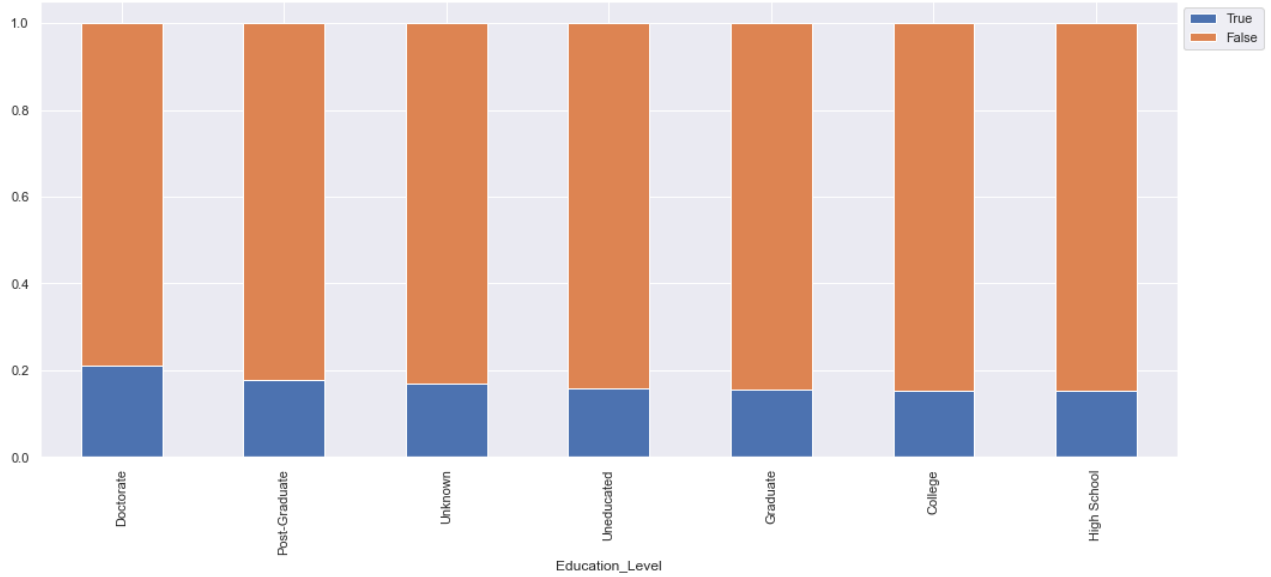


- Proportionaly, there is no significat difference in `Gender` for customer that have Attrition_Flag.

Education_Level vs Attrition_Flag

In [566…
```
stacked_plot(data['Education_Level'])
```

```
Attrition_Flag   True  False    All
Education_Level
All              1627   8500  10127
Graduate          487   2641   3128
High School       306   1707   2013
Unknown           256   1263   1519
Uneducated        237   1250   1487
College           154    859   1013
Doctorate          95    356    451
Post-Graduate      92    424    516
--------------------------------------------------------------------------------
--------------------------------
```

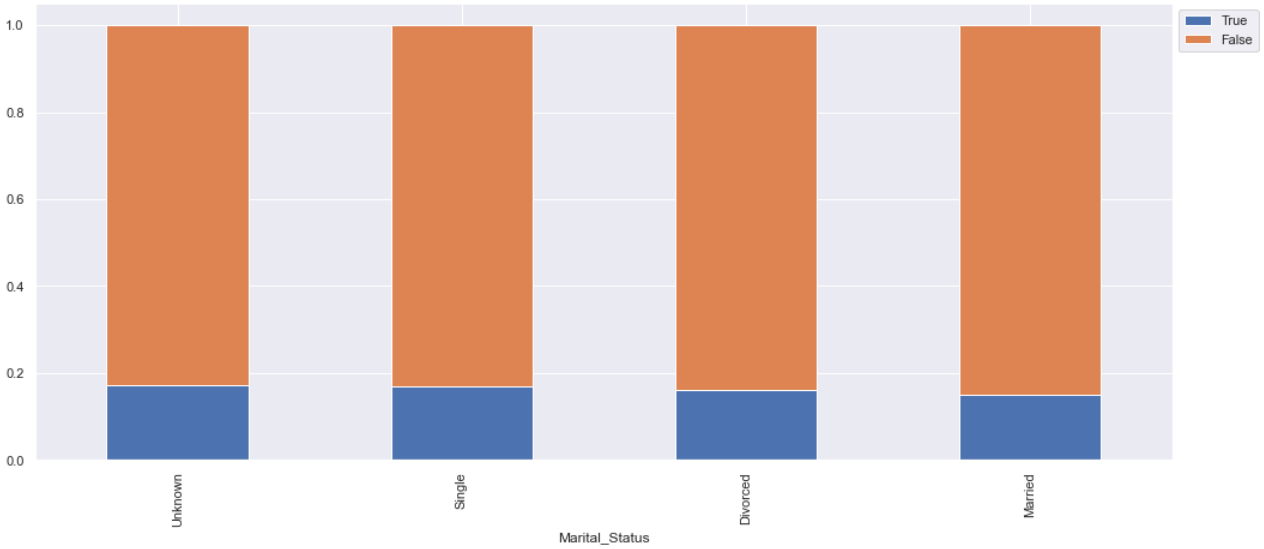


- Proportionaly, there is no significat difference in `Education_Level` for customer that have Attrition_Flag.

Marital_Status vs Attrition_Flag

In [567…
```
stacked_plot(data['Marital_Status'])
```

```
Attrition_Flag  True  False    All
Marital_Status
All             1627   8500  10127
Married          709   3978   4687
Single           668   3275   3943
Unknown          129    620    749
Divorced         121    627    748
------------------------------------------------------------------------------------------
--------------------------------
```

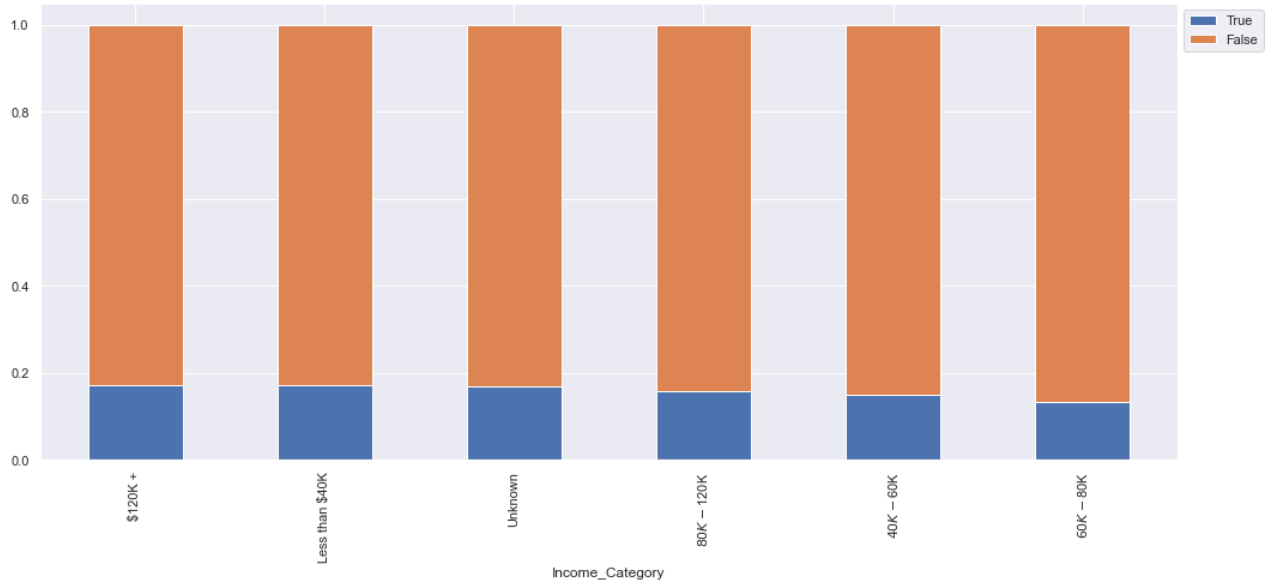


- Proportionaly, there is no significat difference in `Marital_Status` for customer that have Attrition_Flag.

Income_Category vs Attrition_Flag

In [568…
```
stacked_plot(data['Income_Category'])
```

```
Attrition_Flag   True  False    All
Income_Category
All              1627   8500  10127
Less than $40K    612   2949   3561
$40K - $60K       271   1519   1790
$80K - $120K      242   1293   1535
$60K - $80K       189   1213   1402
Unknown           187    925   1112
$120K +           126    601    727
------------------------------------------------------------------------------------------
--------------------------------
```



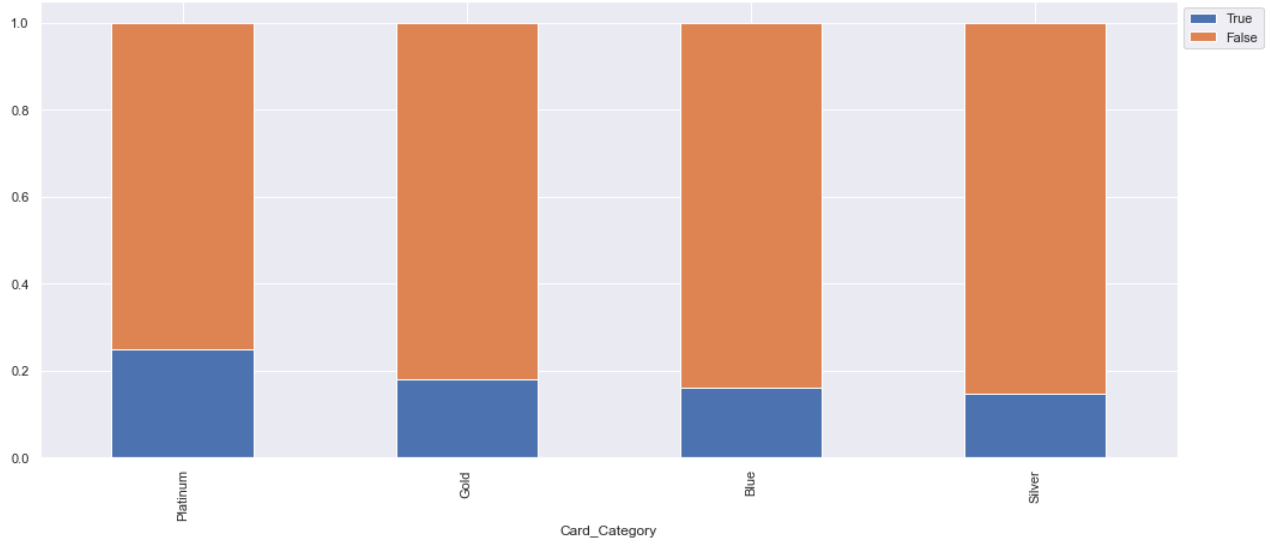
- Proportionaly, there is no significat difference in `Income_Category` for customer that have Attrition_Flag.

Card_Category vs Attrition_Flag

In [569…

```
stacked_plot(data['Card_Category'])
```

```
Attrition_Flag  True  False    All
Card_Category
All             1627   8500  10127
Blue            1519   7917   9436
Silver            82    473    555
Gold              21     95    116
Platinum           5     15     20
--------------------------------------------------------------------------------
--------------------------------
```

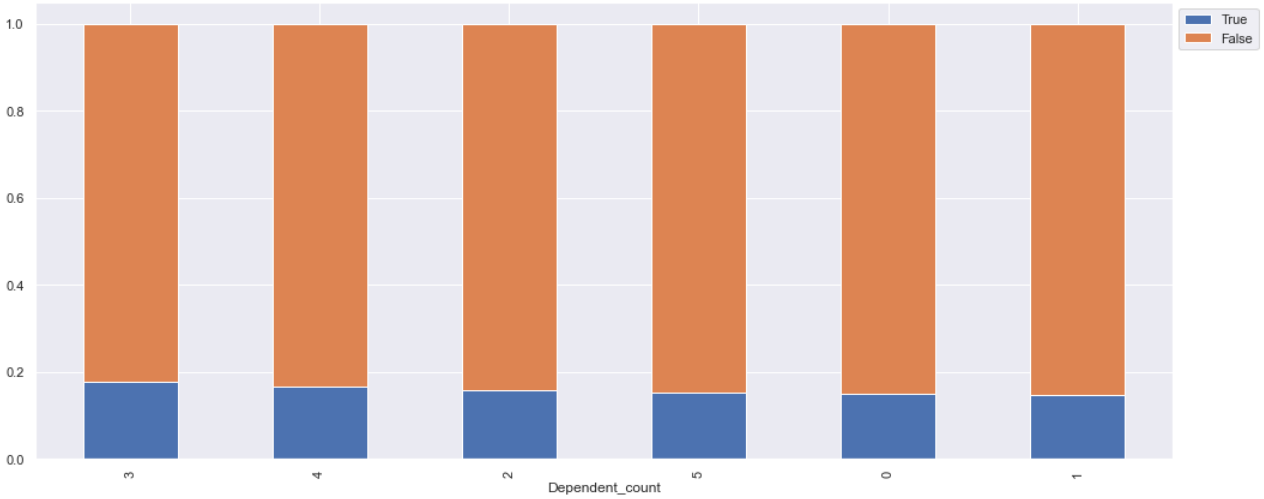


- Proportionaly, there seems to be a slight preference for "Platinum" `Card_Category` when compared to the rest for customer that have Attrition_Flag.

Dependent_count vs Attrition_Flag

In [570…
```
stacked_plot(data['Dependent_count'])
```

```
Attrition_Flag   True  False    All
Dependent_count
All              1627   8500  10127
3                 482   2250   2732
2                 417   2238   2655
1                 269   1569   1838
4                 260   1314   1574
0                 135    769    904
5                  64    360    424
-------------------------------------------------------------------------------------------
--------------------------------
```

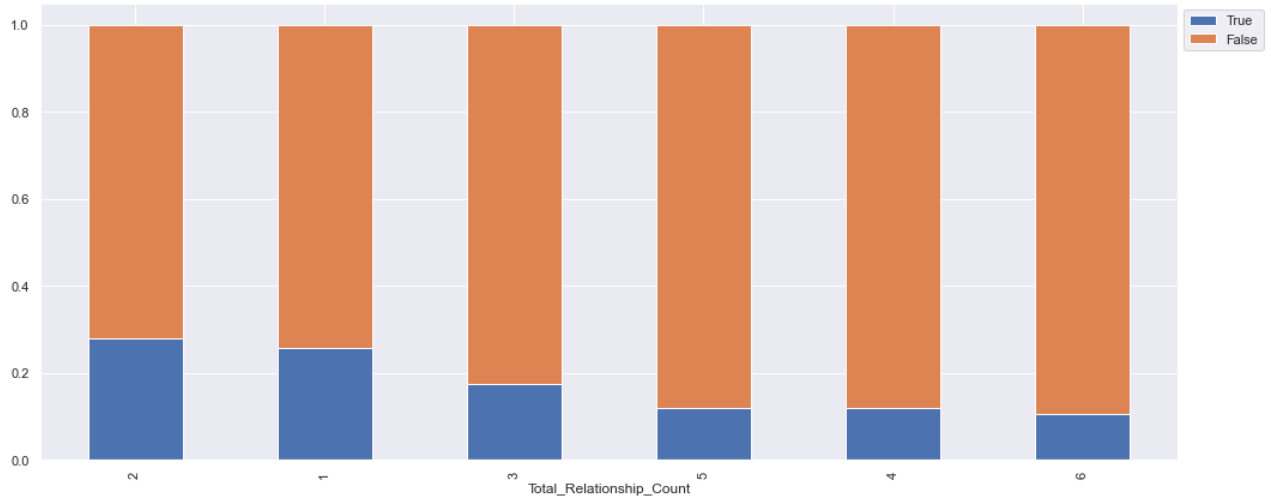


- Proportionaly, there is no significat difference in `Dependent_count` for customer that have Attrition_Flag.

Total_Relationship_Count vs Attrition_Flag

In [571…
```
stacked_plot(data['Total_Relationship_Count'])
```

```
Attrition_Flag            True  False    All
Total_Relationship_Count
All                       1627   8500  10127
3                          400   1905   2305
2                          346    897   1243
1                          233    677    910
5                          227   1664   1891
4                          225   1687   1912
6                          196   1670   1866
-------------------------------------------------------------------------------------------
--------------------------------
```



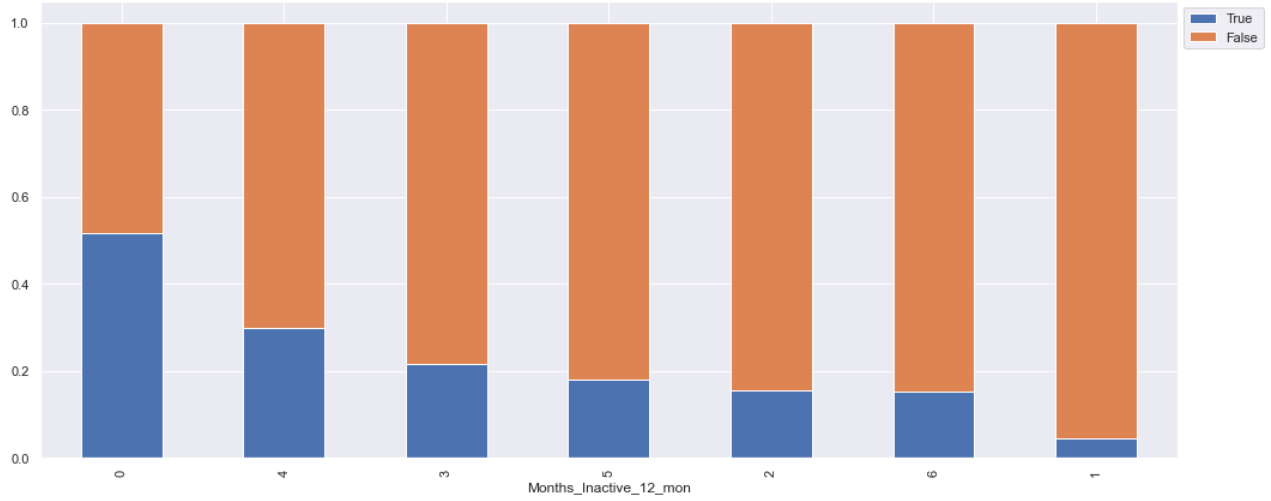
- Proportionaly, there seems to be a slight preference for 1 and 2 `Total_Relationship_Count` when compared to the rest for customer that have Attrition_Flag.

Months_Inactive_12_mon vs Attrition_Flag

In [572…

```
stacked_plot(data['Months_Inactive_12_mon'])
```

```
Attrition_Flag          True  False    All
Months_Inactive_12_mon
All                     1627   8500  10127
3                        826   3020   3846
2                        505   2777   3282
4                        130    305    435
1                        100   2133   2233
5                         32    146    178
6                         19    105    124
0                         15     14     29
------------------------------------------------------------------------------------------
--------------------------------
```

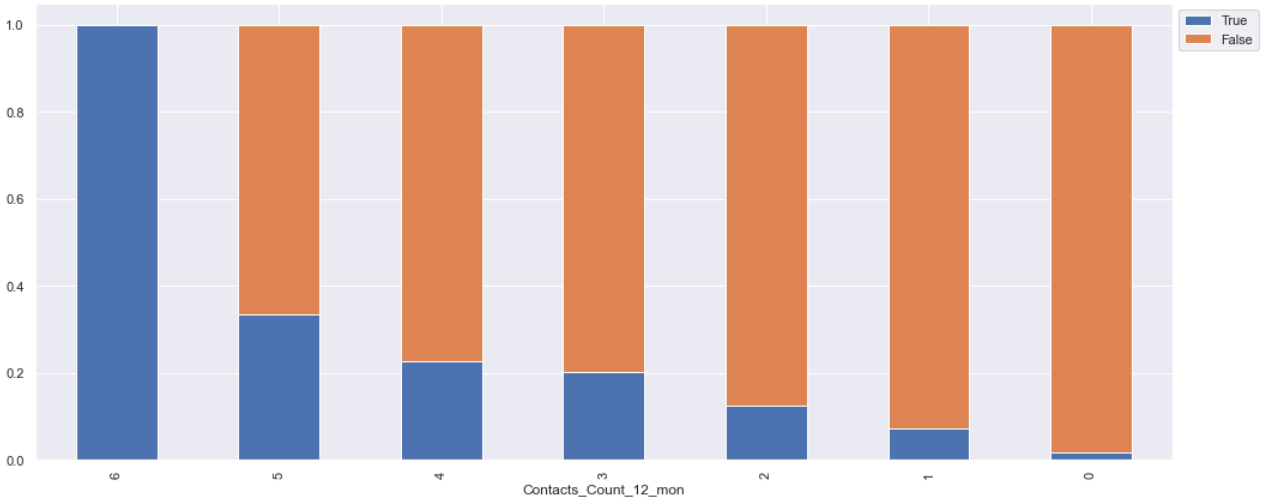


- Proportionaly, there seems to be a slight preference for 0 `Months_Inactive_12_mon` when compared to the rest for customer that have Attrition_Flag.

Contacts_Count_12_mon vs Attrition_Flag

In [573…
```
stacked_plot(data['Contacts_Count_12_mon'])
```

```
Attrition_Flag         True  False    All
Contacts_Count_12_mon
All                    1627   8500  10127
3                       681   2699   3380
2                       403   2824   3227
4                       315   1077   1392
1                       108   1391   1499
5                        59    117    176
6                        54      0     54
0                         7    392    399
------------------------------------------------------------------------------------------
--------------------------------
```



- Proportionaly, there is overwhelming difference for `Contacts_Count_12_mon` with value 6 when compared to the rest for customer that have Attrition_Flag.

In [ ]:
```
```

Multivariate analysis

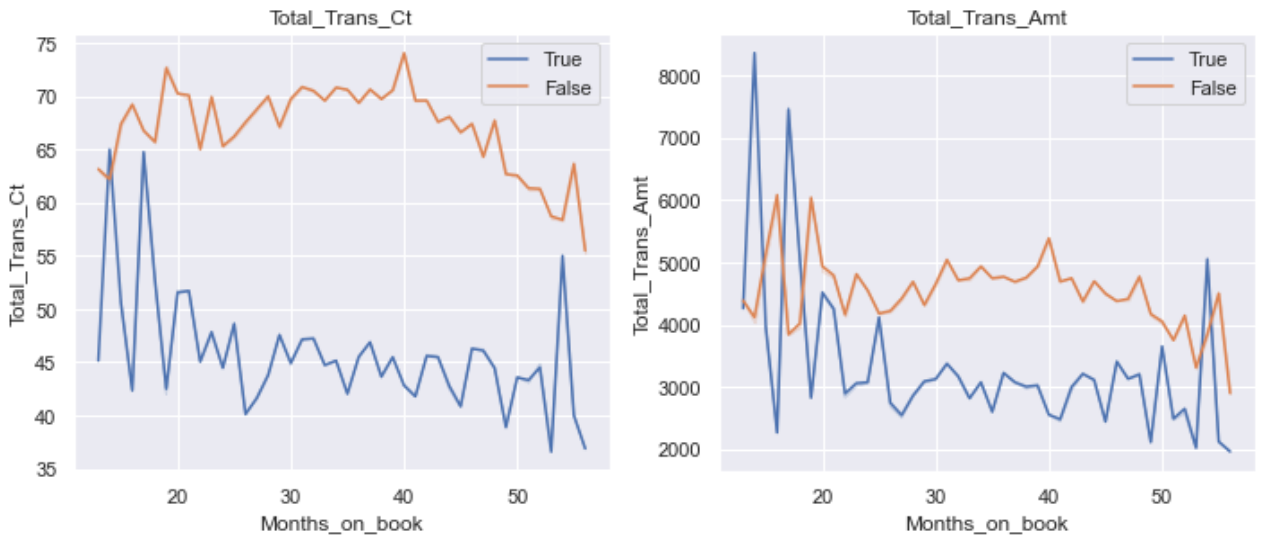
Analys variables with good and high correlation with regards to Attrition_Flag.

- The pairs that have a positive/high (>0.5) correlation are:
  - Credit_limit/Avg_Open_To_Buy
  - Total_Trans_Ct/Total_Trans_Amt
  - Customer_Age/Months_on_book
  - Total_Revolving_Bal/Avg_Utilization_Ratio

In [574…
```
cols = data[['Total_Trans_Ct','Total_Trans_Amt']].columns.tolist()
plt.figure(figsize=(15,12))
for i, variable in enumerate(cols):
                     plt.subplot(3,3,i+1)
                     sns.lineplot(data['Customer_Age'],data[variable],hue=data['Attriti
                     plt.tight_layout()
                     plt.title(variable)
                     plt.legend(bbox_to_anchor=(1, 1))
plt.show()
```

- When visualizing difference bewteen Total_Trans_Ct and Total_Trans_Amt in regards to Customer_Age and Attrition_Flag, there really is no significant difference. This could be a sign of Multicollinearity.

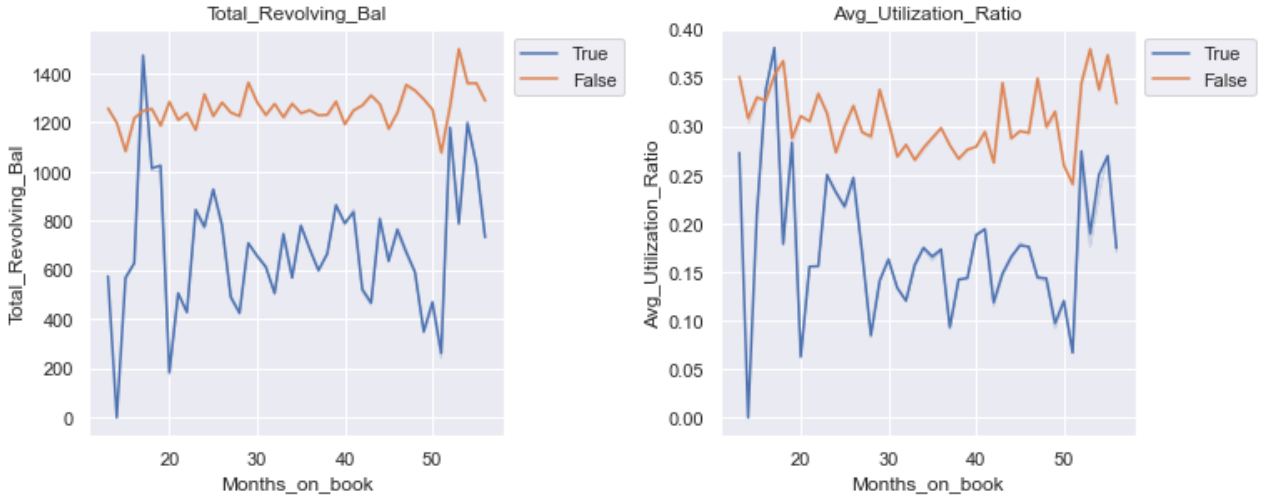
In [575…
```
cols = data[['Total_Trans_Ct','Total_Trans_Amt']].columns.tolist()
plt.figure(figsize=(15,12))
for i, variable in enumerate(cols):
                    plt.subplot(3,3,i+1)
                    sns.lineplot(data['Months_on_book'],data[variable],hue=data['Attri
                    plt.tight_layout()
                    plt.title(variable)
                    plt.legend(bbox_to_anchor=(1, 1))
plt.show()
```



- We can see difference bewteen Total_Trans_Ct and Total_Trans_Amt in regards to Months_on_book and Attrition_Flag.

In [576…
```
cols = data[['Total_Revolving_Bal','Avg_Utilization_Ratio']].columns.tolist()
plt.figure(figsize=(15,12))
for i, variable in enumerate(cols):
                    plt.subplot(3,3,i+1)
                    sns.lineplot(data['Months_on_book'],data[variable],hue=data['Attri
```

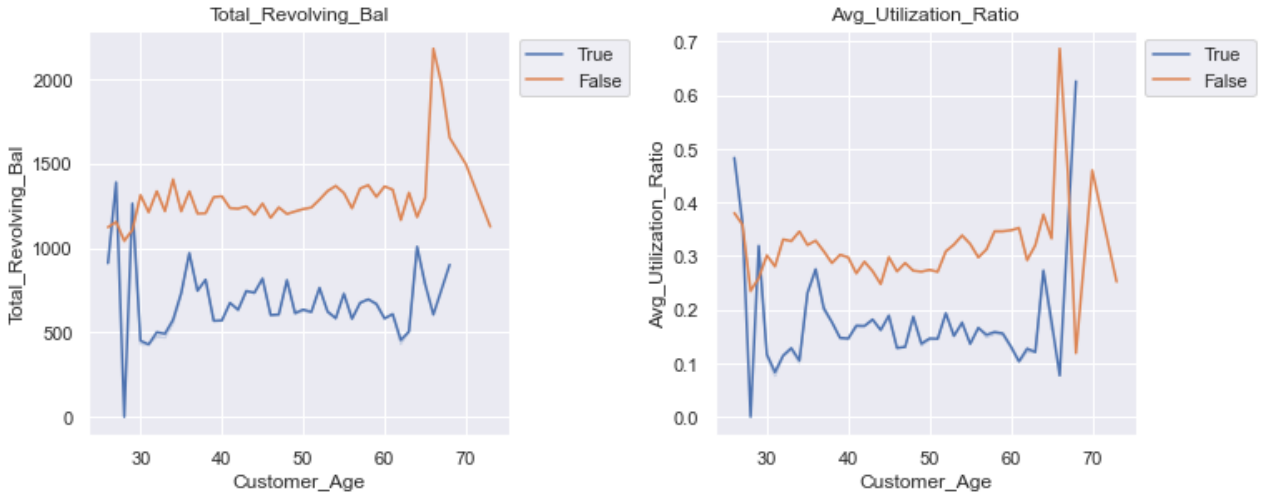
```
                    plt.tight_layout()
                    plt.title(variable)
                    plt.legend(bbox_to_anchor=(1, 1))
plt.show()
```



- When visualizing difference bewteen Total_Revolving_Bal and Avg_Utilization_Ratio in regards to Months_on_book and Attrition_Flag, there really is no significant difference. This could be a sign of Multicollinearity.

In [577…

```
cols = data[['Total_Revolving_Bal','Avg_Utilization_Ratio']].columns.tolist()
plt.figure(figsize=(15,12))
for i, variable in enumerate(cols):
                    plt.subplot(3,3,i+1)
                    sns.lineplot(data['Customer_Age'],data[variable],hue=data['Attriti
                    plt.tight_layout()
                    plt.title(variable)
                    plt.legend(bbox_to_anchor=(1, 1))
plt.show()
```



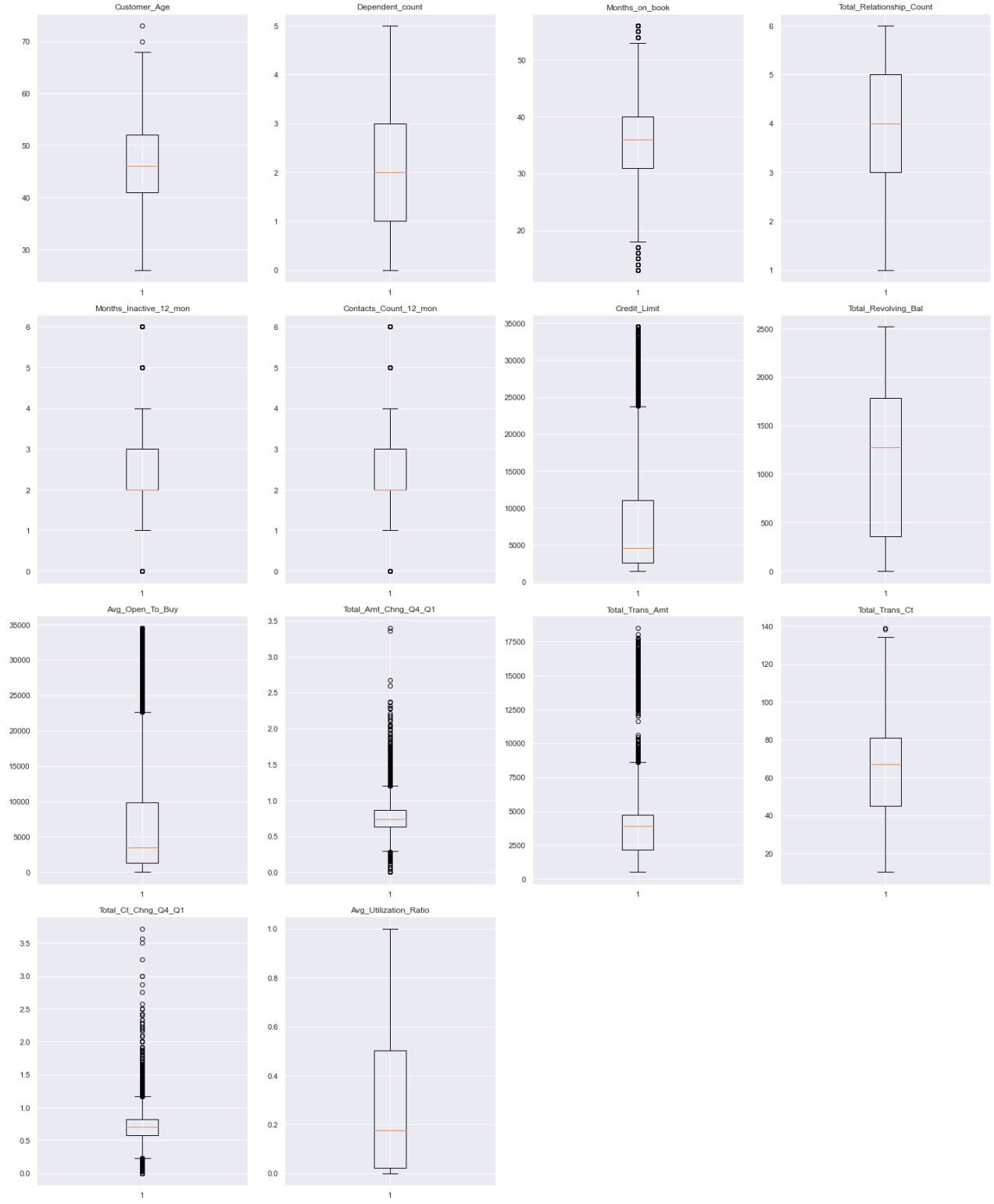
- Between Total_Revolving_Bal and Avg_Utilization_Ratio there are differences in the later Customer_Age(above65) in regards to Attrition_Flag.

Data Pre-processing

Outlier treatment

In [578…

```
numerical_col = data.select_dtypes(include=np.number).columns.tolist()

plt.figure(figsize=(20,30))

for i, variable in enumerate(numerical_col):
                     plt.subplot(5,4,i+1)
                     plt.boxplot(data[variable],whis=1.5)
                     plt.tight_layout()
                     plt.title(variable)

plt.show()
```



- We will remove `Attrition_Flag` from outlier treatment since this is a boolean, and also is the dependant variable.

In [579…

```
def treat_outliers(df,col):
    '''
    treats outliers in a varaible
    col: str, name of the numerical varaible
    df: data frame
    col: name of the column
    '''
```

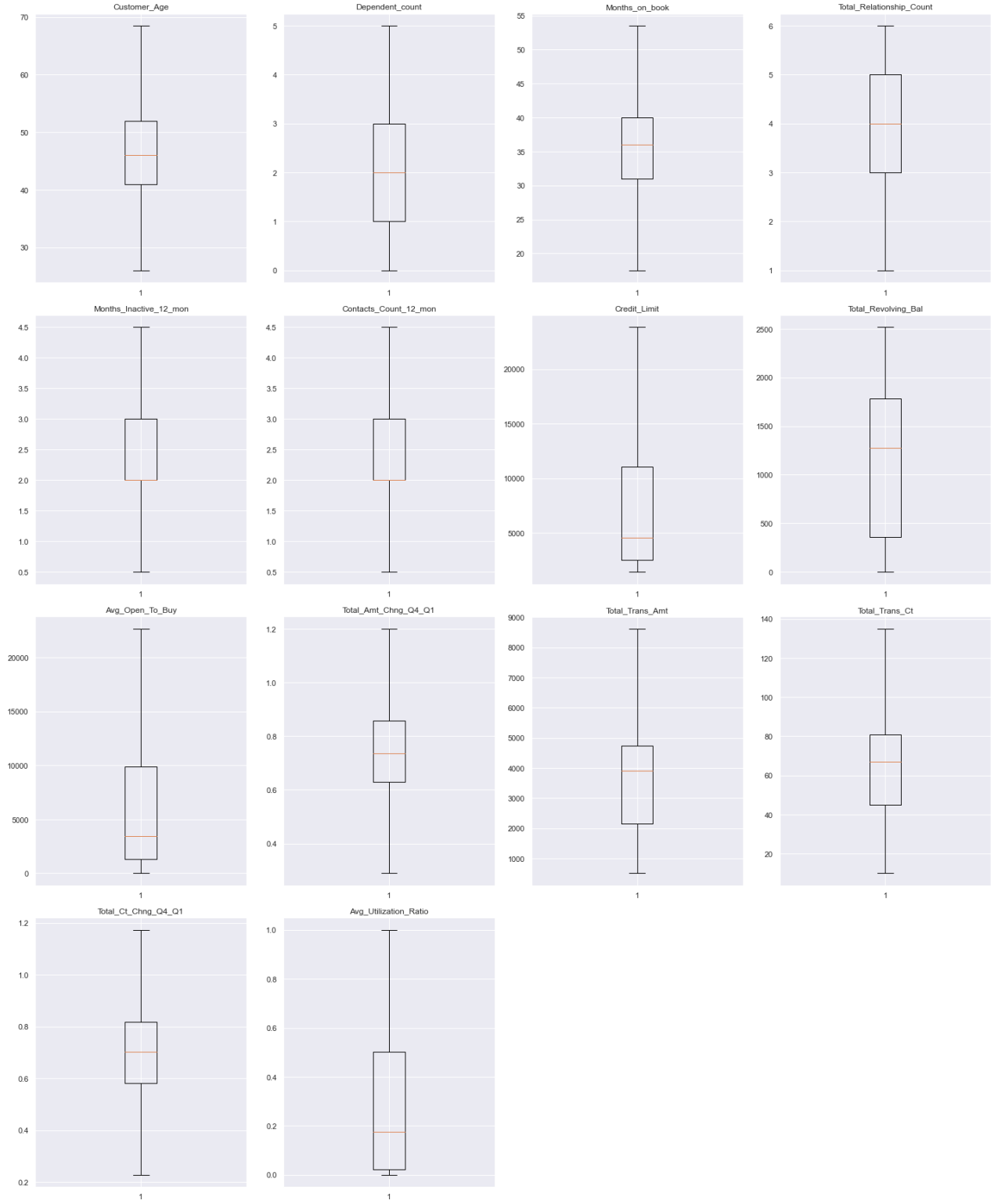
```
    Q1=df[col].quantile(0.25) # 25th quantile
    Q3=df[col].quantile(0.75)  # 75th quantile
    IQR=Q3-Q1
    Lower_Whisker = Q1 - 1.5*IQR
    Upper_Whisker = Q3 + 1.5*IQR
    df[col] = np.clip(df[col], Lower_Whisker, Upper_Whisker) # all the values samller t
                                                             # and all the values above
    return df

def treat_outliers_all(df, col_list):
    '''
    treat outlier in all numerical varaibles
    col_list: list of numerical varaibles
    df: data frame
    '''
    for c in col_list:
        df = treat_outliers(df,c)

    return df
```

In [580…

```
numerical_col = data.select_dtypes(include=np.number).columns.tolist()# getting list of


# items to be removed
unwanted= {'Attrition_Flag'} # these column have very few non zero observation , doing

numerical_col = [ele for ele in numerical_col if ele not in unwanted]
data = treat_outliers_all(data,numerical_col)
```

In [581…

```
numerical_col = data.select_dtypes(include=np.number).columns.tolist()

plt.figure(figsize=(20,30))

for i, variable in enumerate(numerical_col):
                    plt.subplot(5,4,i+1)
                    plt.boxplot(data[variable],whis=1.5)
                    plt.tight_layout()
                    plt.title(variable)

plt.show()
```



- There are no outliers now

Missing-Value Treatment

- We will use KNN imputer to impute missing values.
- We will treat the 'Unknown' values in the following features: Education_Level , Marital_Status , Income_Category

In [582…

```
def convertUnknownValuesToNan(value):
```

```
    if value == 'Unknown' :
        return np.nan
    else:
        return value
```

In [583…
```
data['Education_Level'] = data['Education_Level'].apply(convertUnknownValuesToNan)
data['Marital_Status'] = data['Marital_Status'].apply(convertUnknownValuesToNan)
data['Income_Category'] = data['Income_Category'].apply(convertUnknownValuesToNan)
```

In [584…
```
data.isna().sum()
```

Out[584…
```
Attrition_Flag                0
Customer_Age                  0
Gender                        0
Dependent_count               0
Education_Level            1519
Marital_Status              749
Income_Category            1112
Card_Category                 0
Months_on_book                0
Total_Relationship_Count      0
Months_Inactive_12_mon        0
Contacts_Count_12_mon         0
Credit_Limit                  0
Total_Revolving_Bal           0
Avg_Open_To_Buy               0
Total_Amt_Chng_Q4_Q1          0
Total_Trans_Amt               0
Total_Trans_Ct                0
Total_Ct_Chng_Q4_Q1           0
Avg_Utilization_Ratio         0
dtype: int64
```

KNN Imputer

In [585…
```
imputer = KNNImputer(n_neighbors=5)
reqd_col_for_impute = [ 'Customer_Age','Gender','Education_Level','Marital_Status','Inc
                        'Card_Category','Dependent_count' ]
```

In [586…
```
data[reqd_col_for_impute].head()
```

Out[586…

	Customer_Age	Gender	Education_Level	Marital_Status	Income_Category	Card_Category	Dependent
0	45.0	M	High School	Married	$60K - 80K	Blue	
1	49.0	F	Graduate	Single	Less than $40K	Blue	
2	51.0	M	Graduate	Married	$80K - 120K	Blue	
3	40.0	F	High School	NaN	Less than $40K	Blue	
4	40.0	M	Uneducated	Married	$60K - 80K	Blue	

In [587…
```
data1=data.copy()
```

In [588…
```
# we need to pass numerical values for each categorical column for KNN imputation so we

gender = {'M':0,'F':1}
data1['Gender'] = data1['Gender'].map(gender)
```

```
education_level= {'Graduate':0, 'High School':1,'Uneducated':2,'College':3, 'Post-Gradu
data1['Education_Level'] = data1['Education_Level'].map(education_level)

marital_status = {'Married':0, 'Single':1,'Divorced':2}
data1['Marital_Status'] = data1['Marital_Status'].map(marital_status)

income_category = {'Less than $40K':0,'$40K - $60K':1,'$80K - $120K':2,'$60K - $80K':3,
data1['Income_Category'] = data1['Income_Category'].map(income_category)

card_category ={'Blue':0,'Silver':1,'Gold':2,'Platinum':3 }
data1['Card_Category'] = data1['Card_Category'].map(card_category)
```

In [589…
```
data1.head()
```

Out[589…

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Cat
0	False	45.0	0	3	1.0	0.0	
1	False	49.0	1	5	0.0	1.0	
2	False	51.0	0	3	0.0	0.0	
3	False	40.0	1	4	1.0	NaN	
4	False	40.0	0	3	2.0	0.0	

- Values have been encoded.

Split Data

In [590…
```
X = data1.drop(['Attrition_Flag'],axis=1)
y = data1['Attrition_Flag'].apply(lambda x : 1 if x==True else 0)

# Splitting data into training and test set:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=7
print(X_train.shape, X_test.shape)
```

```
(7088, 19) (3039, 19)
```

Imputing Missing Values

In [591…
```
#Fit and transform the train data
X_train[reqd_col_for_impute]=imputer.fit_transform(X_train[reqd_col_for_impute])

#Transform the test data
X_test[reqd_col_for_impute]=imputer.transform(X_test[reqd_col_for_impute])

#Checking that no column has missing values in train or test sets
print(X_train.isna().sum())
print('-'*30)
print(X_test.isna().sum())
```

```
Customer_Age              0
Gender                    0
Dependent_count           0
Education_Level           0
Marital_Status            0
Income_Category           0
```

```
Card_Category               0
Months_on_book              0
Total_Relationship_Count    0
Months_Inactive_12_mon      0
Contacts_Count_12_mon       0
Credit_Limit                0
Total_Revolving_Bal         0
Avg_Open_To_Buy             0
Total_Amt_Chng_Q4_Q1        0
Total_Trans_Amt             0
Total_Trans_Ct              0
Total_Ct_Chng_Q4_Q1         0
Avg_Utilization_Ratio       0
dtype: int64
------------------------------
Customer_Age                0
Gender                      0
Dependent_count             0
Education_Level             0
Marital_Status              0
Income_Category             0
Card_Category               0
Months_on_book              0
Total_Relationship_Count    0
Months_Inactive_12_mon      0
Contacts_Count_12_mon       0
Credit_Limit                0
Total_Revolving_Bal         0
Avg_Open_To_Buy             0
Total_Amt_Chng_Q4_Q1        0
Total_Trans_Amt             0
Total_Trans_Ct              0
Total_Ct_Chng_Q4_Q1         0
Avg_Utilization_Ratio       0
dtype: int64
```

- All 'Unknown' values have been treated.
- Now, we must inverse map the encoded values.

In [592…
```
## Function to inverse the encoding
def inverse_mapping(x,y):
    inv_dict = {v: k for k, v in x.items()}
    X_train[y] = np.round(X_train[y]).map(inv_dict).astype('category')
    X_test[y] = np.round(X_test[y]).map(inv_dict).astype('category')
```

In [593…
```
inverse_mapping(gender,'Gender')
inverse_mapping(education_level,'Education_Level')
inverse_mapping(marital_status,'Marital_Status')
inverse_mapping(income_category,'Income_Category')
inverse_mapping(card_category,'Card_Category')
```

- Checking inverse mapped values/categories.

In [594…
```
cols = X_train.select_dtypes(include=['object','category'])
for i in cols.columns:
    print(X_train[i].value_counts())
    print('*'*30)
```

```
F    3753
M    3335
Name: Gender, dtype: int64
```

```
******************************
Graduate         2257
High School      1909
Uneducated       1472
College           770
Post-Graduate     358
Doctorate         322
Name: Education_Level, dtype: int64
******************************
Married     3456
Single      3094
Divorced     538
Name: Marital_Status, dtype: int64
******************************
Less than $40K    3055
$40K - $60K       1443
$80K - $120K      1127
$60K - $80K        959
$120K +            504
Name: Income_Category, dtype: int64
******************************
Blue        6619
Silver       373
Gold          81
Platinum      15
Name: Card_Category, dtype: int64
******************************
```

In [595…

```
cols = X_test.select_dtypes(include=['object','category'])
for i in cols.columns:
    print(X_test[i].value_counts())
    print('*'*30)
```

```
F    1605
M    1434
Name: Gender, dtype: int64
******************************
Graduate         947
High School      858
Uneducated       621
College          320
Post-Graduate    164
Doctorate        129
Name: Education_Level, dtype: int64
******************************
Married     1508
Single      1321
Divorced     210
Name: Marital_Status, dtype: int64
******************************
Less than $40K    1318
$40K - $60K        613
$60K - $80K        450
$80K - $120K       435
$120K +            223
Name: Income_Category, dtype: int64
******************************
Blue        2817
Silver       182
Gold          35
Platinum       5
Name: Card_Category, dtype: int64
******************************
```

- Inverse mapping returned original labels.

Encoding categorical varaibles

In [596…
```
X_train=pd.get_dummies(X_train,drop_first=True)
X_test=pd.get_dummies(X_test,drop_first=True)
print(X_train.shape, X_test.shape)
```

```
(7088, 29) (3039, 29)
```

- After encoding there are 29 columns.

Building the model

Model evaluation criterion:

Model can make wrong predictions as:

- Predicting an customer will attrite , but in reality, the customer will not attrite. - Loss of opportunity
- Predicting an customer will not attrite, but the customer does attrite - Loss of resources

Most importantly,

- Predicting an customer will attrite, but the enrollee does not want to i.e. losing on a potential business as the customer will not be targeted by the Sales team when he should be targeted.

In order to reduce this loss, we need to reduce False Negatives

- Company would want Recall to be maximized, greater the Recall lesser the chances of false negatives.

In [597…
```
##  Function to calculate different metric scores of the model - Accuracy, Recall and P
def get_metrics_score(model,train,test,train_y,test_y,flag=True):
    '''
    model : classifier to predict values of X

    '''
    # defining an empty list to store train and test results
    score_list=[]

    pred_train = model.predict(train)
    pred_test = model.predict(test)

    train_acc = model.score(train,train_y)
    test_acc = model.score(test,test_y)

    train_recall = metrics.recall_score(train_y,pred_train)
    test_recall = metrics.recall_score(test_y,pred_test)

    train_precision = metrics.precision_score(train_y,pred_train)
    test_precision = metrics.precision_score(test_y,pred_test)

    score_list.extend((train_acc,test_acc,train_recall,test_recall,train_precision,test

    # If the flag is set to True then only the following print statements will be displ
```

```
    if flag == True:
        print("Accuracy on training set : ",model.score(train,train_y))
        print("Accuracy on test set : ",model.score(test,test_y))
        print("Recall on training set : ",metrics.recall_score(train_y,pred_train))
        print("Recall on test set : ",metrics.recall_score(test_y,pred_test))
        print("Precision on training set : ",metrics.precision_score(train_y,pred_train
        print("Precision on test set : ",metrics.precision_score(test_y,pred_test))

    return score_list # returning the list with train and test scores
```

In [598…
```
def make_confusion_matrix(model,y_actual,labels=[1, 0]):
    '''
    model : classifier to predict values of X
    y_actual : ground truth

    '''
    y_predict = model.predict(X_test)
    cm=metrics.confusion_matrix( y_actual, y_predict, labels=[0, 1])
    df_cm = pd.DataFrame(cm, index = [i for i in ["Actual - No","Actual - Yes"]],
                         columns = [i for i in ['Predicted - No','Predicted - Yes']])
    group_counts = ["{0:0.0f}".format(value) for value in
                    cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in
                         cm.flatten()/np.sum(cm)]
    labels = [f"{v1}\n{v2}" for v1, v2 in
              zip(group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    plt.figure(figsize = (10,7))
    sns.heatmap(df_cm, annot=labels,fmt='')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [ ]:
```
```

- Finally, let convert ProdTaken to category so it is handled correctly

Logistic Regression

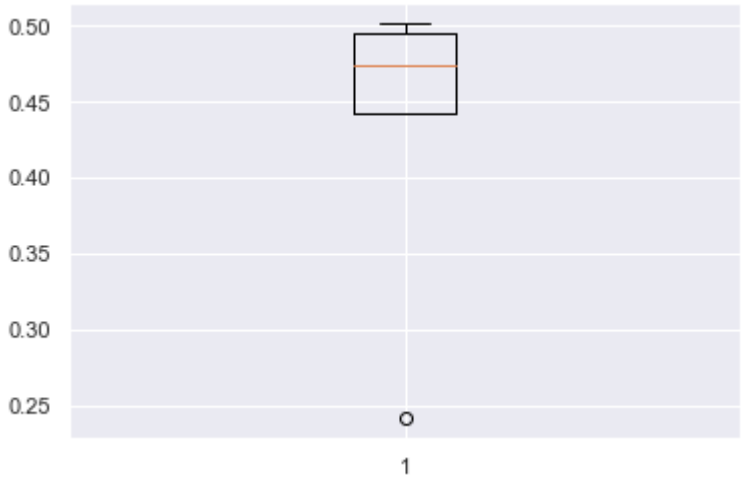
In [599…
```
lr = LogisticRegression(random_state=1)
lr.fit(X_train,y_train)
```

Out[599… `LogisticRegression(random_state=1)`

Evaluate the model performance by using KFold and cross_val_score

- K-Folds cross-validation provides dataset indices to split data into train/validation sets. Split dataset into k consecutive stratified folds (without shuffling by default). Each fold is then used once as validation while the k - 1 remaining folds form the training set.

In [600…
```
scoring='recall'
kfold=StratifiedKFold(n_splits=5,shuffle=True,random_state=1)     #Setting number of sp
cv_result_bfr=cross_val_score(estimator=lr, X=X_train, y=y_train, scoring=scoring, cv=k
#Plotting boxplots for CV scores of model defined above
plt.boxplot(cv_result_bfr)
plt.show()
```

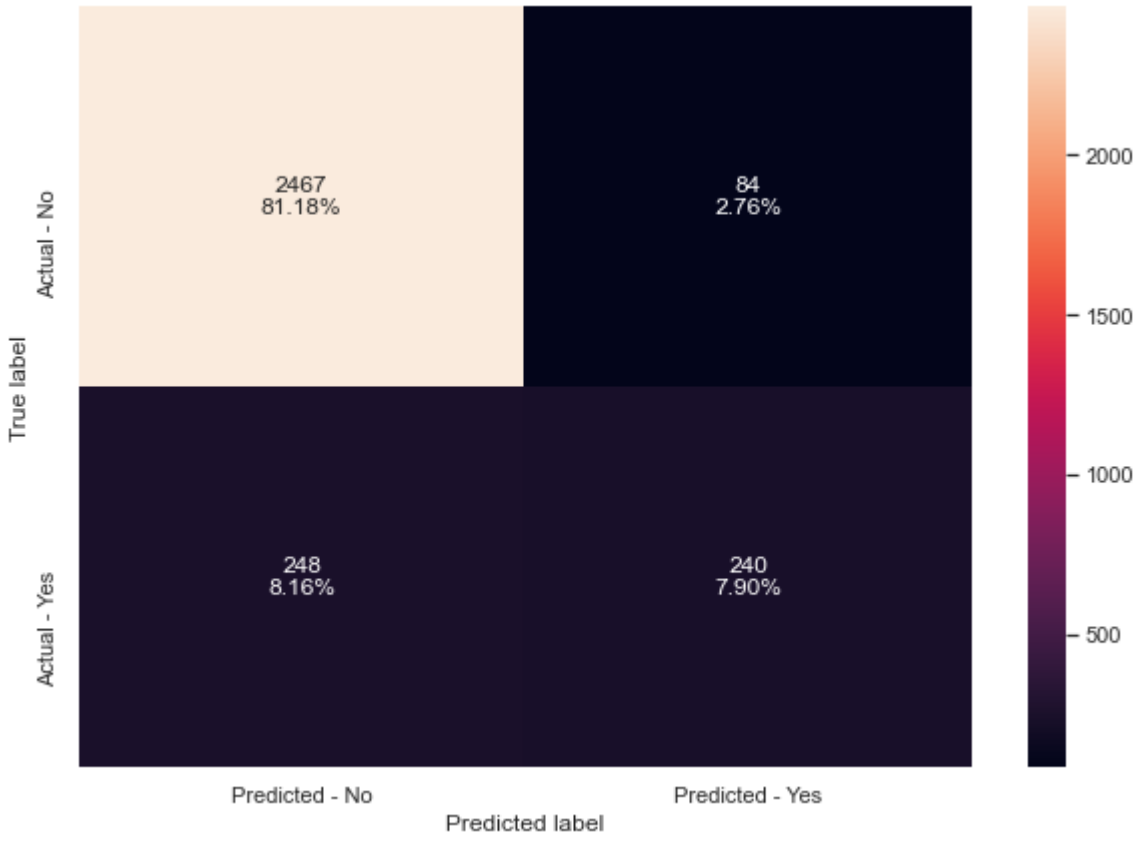


- Performance on training set varies between 0.25 to 0.50 recall.
- Let's check the performance on test data.

In [601…

```
#Calculating different metrics
scores_LR = get_metrics_score(lr,X_train,X_test,y_train,y_test)

# creating confusion matrix
make_confusion_matrix(lr,y_test)
```

```
Accuracy on training set :  0.8844525959367946
Accuracy on test set :  0.8907535373478118
Recall on training set :  0.47058823529411764
Recall on test set :  0.4918032786885246
Precision on training set :  0.7127659574468085
Precision on test set :  0.7407407407407407
```



- Logistic Regression has given a generalized performance on training and test set.
- Recall is very low, we can try oversampling (increase training data) to see if the model performance can be improved.

Oversampling train data using SMOTE

In [602…
```
from imblearn.over_sampling import SMOTE
```

In [603…
```
print("Before UpSampling, counts of label 'Yes': {}".format(sum(y_train==1)))
print("Before UpSampling, counts of label 'No': {} \n".format(sum(y_train==0)))

sm = SMOTE(sampling_strategy = 1 ,k_neighbors = 5, random_state=1)   #Synthetic Minorit
X_train_over, y_train_over = sm.fit_resample(X_train, y_train)


print("After UpSampling, counts of label 'Yes': {}".format(sum(y_train_over==1)))
print("After UpSampling, counts of label 'No': {} \n".format(sum(y_train_over==0)))


print('After UpSampling, the shape of train_X: {}'.format(X_train_over.shape))
print('After UpSampling, the shape of train_y: {} \n'.format(y_train_over.shape))
```

```
Before UpSampling, counts of label 'Yes': 1139
Before UpSampling, counts of label 'No': 5949

After UpSampling, counts of label 'Yes': 5949
After UpSampling, counts of label 'No': 5949

After UpSampling, the shape of train_X: (11898, 29)
After UpSampling, the shape of train_y: (11898,)
```

Logistic Regression on oversampled data

In [604…
```
log_reg_over = LogisticRegression(random_state = 1)

# Training the basic logistic regression model with training set
log_reg_over.fit(X_train_over,y_train_over)
```

Out[604…
```
LogisticRegression(random_state=1)
```

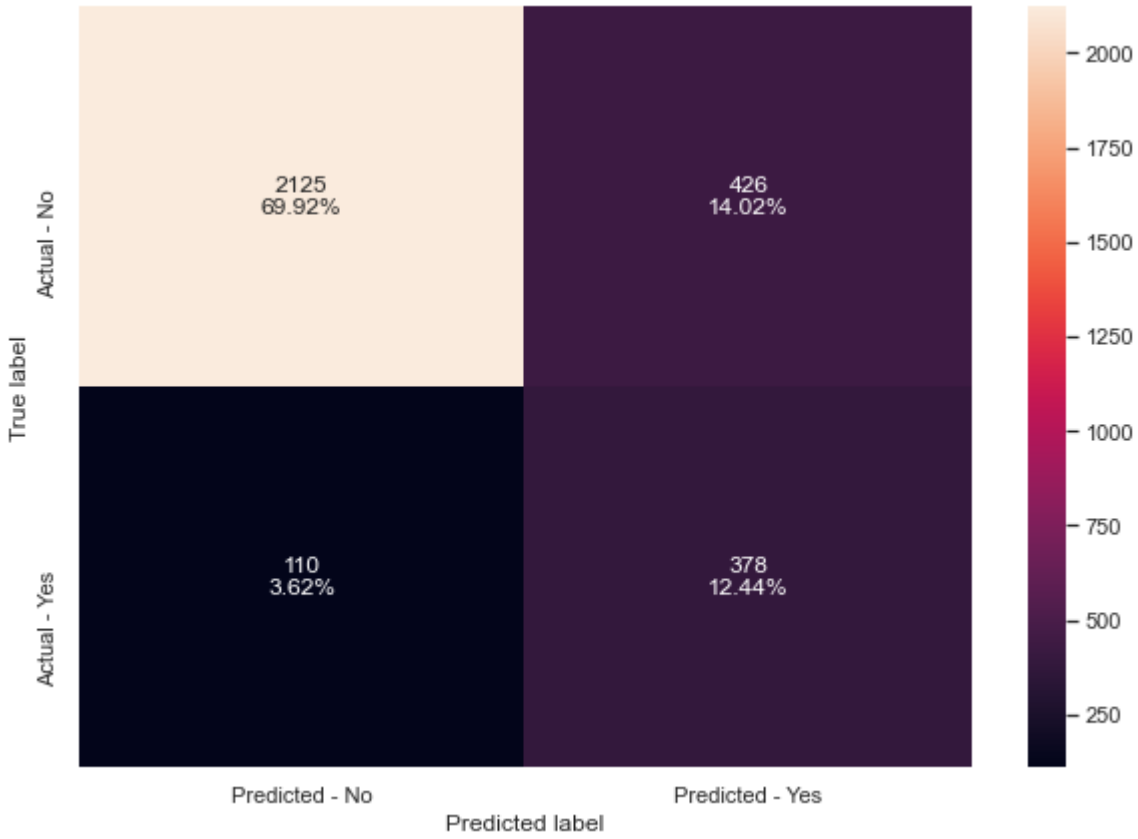
In [605…
```
scoring='recall'
kfold=StratifiedKFold(n_splits=5,shuffle=True,random_state=1)     #Setting number of sp
cv_result_over=cross_val_score(estimator=log_reg_over, X=X_train_over, y=y_train_over,
#Plotting boxplots for CV scores of model defined above
plt.boxplot(cv_result_over)
plt.show()
```

- Performance of model on training set varies between 0.825 to 0.850, which is an improvement from the previous model
- Let's check the performance on the test set.

In [606…

```
#Calculating different metrics
get_metrics_score(log_reg_over,X_train_over,X_test,y_train_over,y_test)

# creating confusion matrix
make_confusion_matrix(log_reg_over,y_test)
```

```
Accuracy on training set :  0.83761976802824
Accuracy on test set :  0.8236261928265877
Recall on training set :  0.8411497730711044
Recall on test set :  0.7745901639344263
Precision on training set :  0.8352528793189785
Precision on test set :  0.4701492537313433
```



- Performance on the training set improved but the model is not able to replicate the same for the test set.
- Model is overfitting.
- Lets try:

  a) Regularization to see if overfitting can be reduced

  b) Undersampling the train to handle the imbalance between classes and check the model performance.

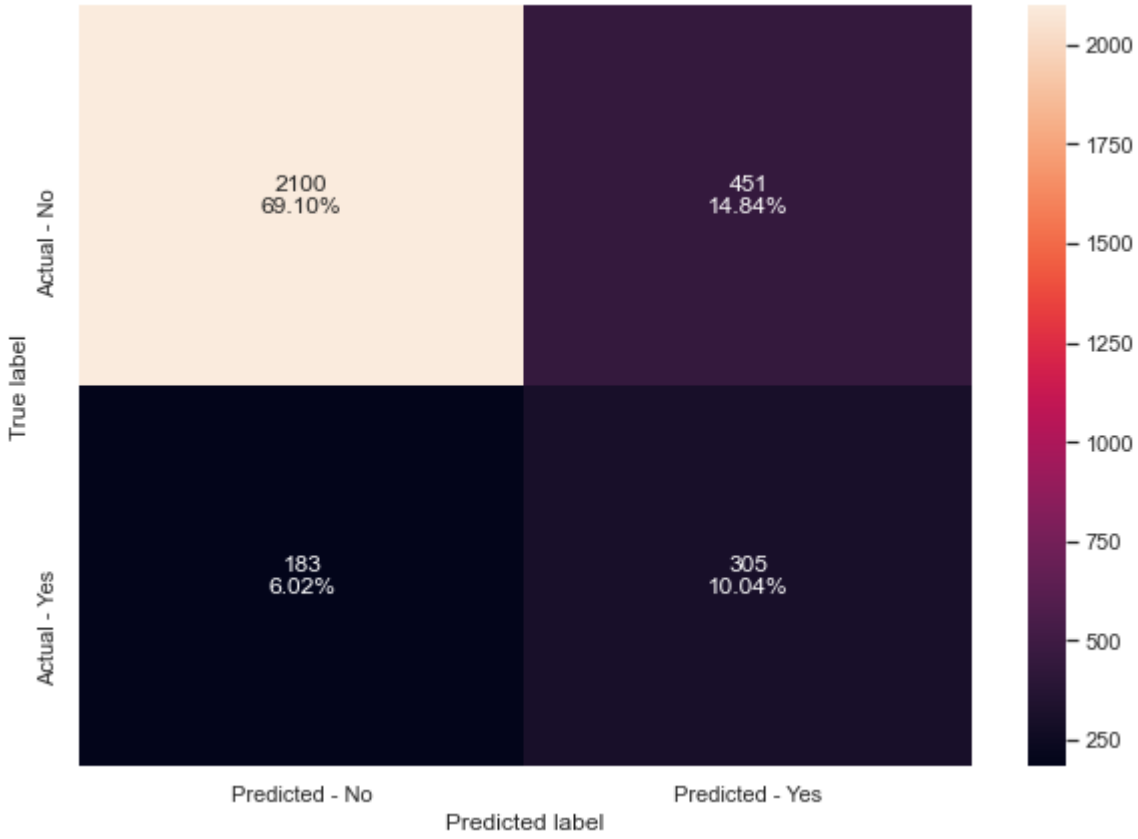
Regularization

In [607…
```
# Choose the type of classifier.
lr_estimator = LogisticRegression(random_state=1,solver='saga')

# Grid of parameters to choose from
parameters = {'C': np.arange(0.1,1.1,0.1)}

# Run the grid search
grid_obj = GridSearchCV(lr_estimator, parameters, scoring='recall')
grid_obj = grid_obj.fit(X_train_over, y_train_over)

# Set the clf to the best combination of parameters
lr_estimator = grid_obj.best_estimator_

# Fit the best algorithm to the data.
lr_estimator.fit(X_train_over, y_train_over)
```

Out[607…
```
LogisticRegression(C=0.1, random_state=1, solver='saga')
```

In [608…
```
#Calculating different metrics
get_metrics_score(lr_estimator,X_train_over,X_test,y_train_over,y_test)

# creating confusion matrix
make_confusion_matrix(lr_estimator,y_test)
```

```
Accuracy on training set :  0.7254160363086233
Accuracy on test set :  0.7913787430075683
Recall on training set :  0.6396032946713733
Recall on test set :  0.625
Precision on training set :  0.7721185064935064
Precision on test set :  0.40343915343915343
```



- After regularization, overfitting has reduced to some extent and the model is also performing well.

Undersampling train data using SMOTE

In [609…
```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state = 1)
X_train_un, y_train_un = rus.fit_resample(X_train, y_train)
```

In [610…
```
print("Before Under Sampling, counts of label 'Yes': {}".format(sum(y_train==1)))
print("Before Under Sampling, counts of label 'No': {} \n".format(sum(y_train==0)))

print("After Under Sampling, counts of label 'Yes': {}".format(sum(y_train_un==1)))
print("After Under Sampling, counts of label 'No': {} \n".format(sum(y_train_un==0)))

print('After Under Sampling, the shape of train_X: {}'.format(X_train_un.shape))
print('After Under Sampling, the shape of train_y: {} \n'.format(y_train_un.shape))
```

```
Before Under Sampling, counts of label 'Yes': 1139
Before Under Sampling, counts of label 'No': 5949

After Under Sampling, counts of label 'Yes': 1139
After Under Sampling, counts of label 'No': 1139

After Under Sampling, the shape of train_X: (2278, 29)
After Under Sampling, the shape of train_y: (2278,)
```

Logistic Regression on undersampled data

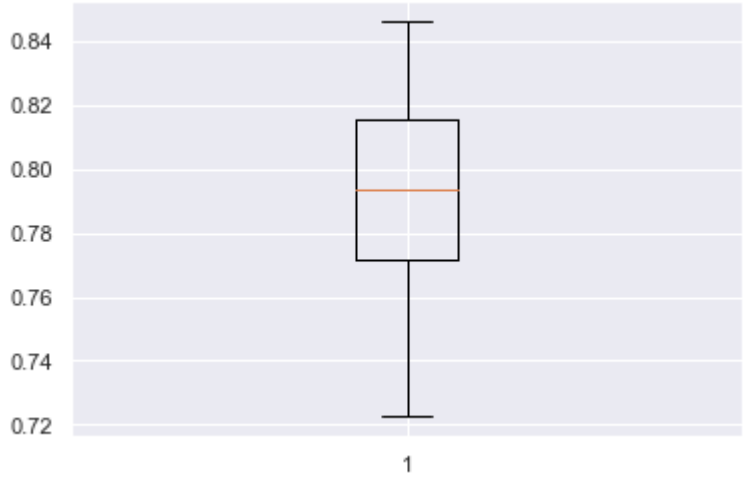
Let's evaluate the model performance by using KFold and cross_val_score

- K-Folds cross-validation provides dataset indices to split data into train/validation sets. Split dataset into k consecutive stratified folds (without shuffling by default). Each fold is then used once as validation while the k - 1 remaining folds form the training set.

In [611…
```
log_reg_under = LogisticRegression(random_state = 1)
log_reg_under.fit(X_train_un,y_train_un )
```

Out[611…
```
LogisticRegression(random_state=1)
```

In [612…
```
scoring='recall'
kfold=StratifiedKFold(n_splits=5,shuffle=True,random_state=1)     #Setting number of sp
cv_result_under=cross_val_score(estimator=log_reg_under, X=X_train_un, y=y_train_un, sc
#Plotting boxplots for CV scores of model defined above
plt.boxplot(cv_result_under)
plt.show()
```

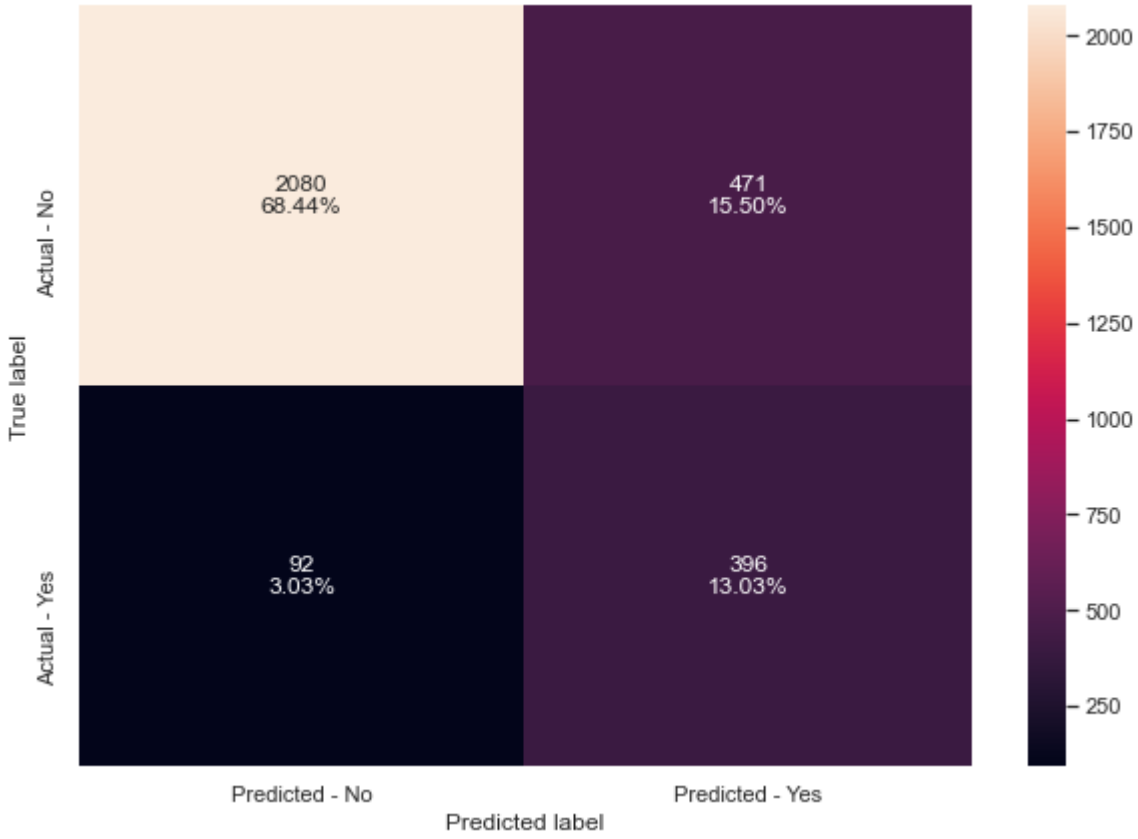


- Performance of model on training set varies between 0.72 to 0.85, which is an improvement from the initial model(without oversampling)

Performance on the test set

In [613…
```
#Calculating different metrics
get_metrics_score(log_reg_under,X_train_un,X_test,y_train_un,y_test)


# creating confusion matrix
make_confusion_matrix(log_reg_under,y_test)
```

```
Accuracy on training set :  0.810359964881475
Accuracy on test set :  0.8147416913458374
Recall on training set :  0.8121158911325724
Recall on test set :  0.8114754098360656
Precision on training set :  0.8092738407699037
Precision on test set :  0.45674740484429066
```



- Model has given a generalized performance on training and test set.
- Model performance has improved using downsampling - Logistic regression is now able to differentiate well between positive and negative classes.

In [614…

```
# defining list of model
models = [lr]

# defining empty lists to add train and test results
acc_train = []
acc_test = []
recall_train = []
recall_test = []
precision_train = []
precision_test = []

# looping through all the models to get the metrics score - Accuracy, Recall and Precis
for model in models:

    j = get_metrics_score(model,X_train,X_test,y_train,y_test,False)
    acc_train.append(j[0])
    acc_test.append(j[1])
    recall_train.append(j[2])
    recall_test.append(j[3])
    precision_train.append(j[4])
    precision_test.append(j[5])
```

In [615…

```
# defining list of models
models = [log_reg_over, lr_estimator]

# looping through all the models to get the metrics score - Accuracy, Recall and Precis
for model in models:
```

```
    j = get_metrics_score(model,X_train_over,X_test,y_train_over,y_test,False)
    acc_train.append(j[0])
    acc_test.append(j[1])
    recall_train.append(j[2])
    recall_test.append(j[3])
    precision_train.append(j[4])
    precision_test.append(j[5])
```

In [616…

```
# defining list of model
models = [log_reg_under]

# looping through all the models to get the metrics score - Accuracy, Recall and Precis
for model in models:

    j = get_metrics_score(model,X_train_un,X_test,y_train_un,y_test,False)
    acc_train.append(j[0])
    acc_test.append(j[1])
    recall_train.append(j[2])
    recall_test.append(j[3])
    precision_train.append(j[4])
    precision_test.append(j[5])
```

In [617…

```
comparison_frame = pd.DataFrame({'Model':['Logistic Regression','Logistic Regression on
                                          'Logistic Regression-Regularized (Oversampled
                                          'Train_Accuracy': acc_train,'Test_Accuracy':
                                          'Train_Recall':recall_train,'Test_Recall':rec
                                          'Train_Precision':precision_train,'Test_Preci

#Sorting models in decreasing order of test recall
comparison_frame
```

Out[617…

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision
0	Logistic Regression	0.884453	0.890754	0.470588	0.491803	0.712766	0.740741
1	Logistic Regression on Oversampled data	0.837620	0.823626	0.841150	0.774590	0.835253	0.470149
2	Logistic Regression-Regularized (Oversampled d...	0.725416	0.791379	0.639603	0.625000	0.772119	0.403439
3	Logistic Regression on Undersampled data	0.810360	0.814742	0.812116	0.811475	0.809274	0.456747

- `Logistic Regression on Undersampled data` has given a generalized performance with the highest recall on test data (0.811475).

In [ ]:

Building different models using KFold and cross_val_score with pipelines and tune the best model using GridSearchCV and RandomizedSearchCV

- `Stratified K-Folds cross-validation` provides dataset indices to split data into train/validation sets. Split dataset into k consecutive folds (without shuffling by default) keeping the distribution of both classes in each fold the same as the target variable. Each fold is then used once as validation while the k - 1 remaining folds form the training set.

In [618…

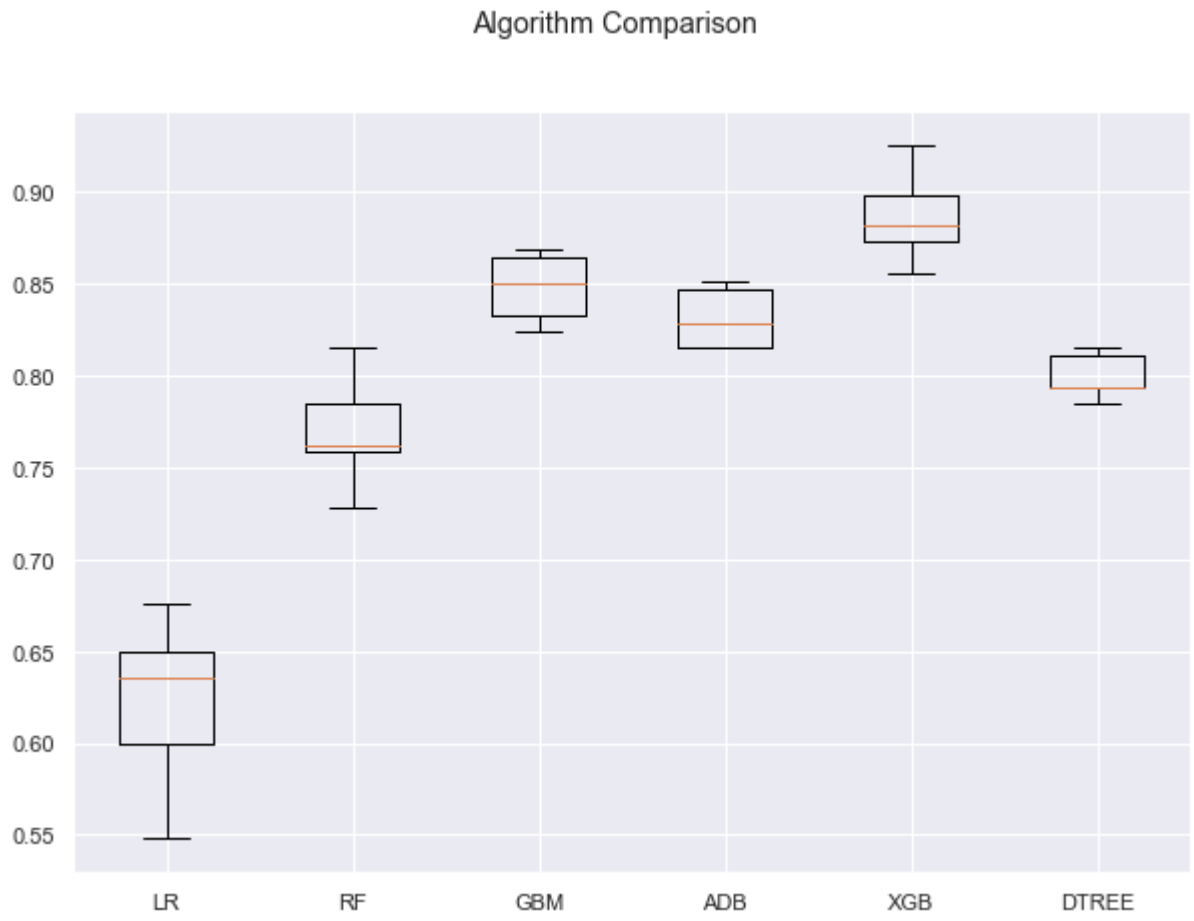
```
models = []  # Empty list to store all the models

# Appending pipelines for each model into the list
models.append(
    (
        "LR",
        Pipeline(
            steps=[
                ("scaler", StandardScaler()),
                ("log_reg", LogisticRegression(random_state=1)),
            ]
        ),
    )
)
models.append(
    (
        "RF",
        Pipeline(
            steps=[
                ("scaler", StandardScaler()),
                ("random_forest", RandomForestClassifier(random_state=1)),
            ]
        ),
    )
)
models.append(
    (
        "GBM",
        Pipeline(
            steps=[
                ("scaler", StandardScaler()),
                ("gradient_boosting", GradientBoostingClassifier(random_state=1)),
            ]
        ),
    )
)
models.append(
    (
        "ADB",
        Pipeline(
            steps=[
                ("scaler", StandardScaler()),
                ("adaboost", AdaBoostClassifier(random_state=1)),
            ]
        ),
    )
)
models.append(
    (
```

```
        "XGB",
        Pipeline(
            steps=[
                ("scaler", StandardScaler()),
                ("xgboost", XGBClassifier(random_state=1,eval_metric='logloss')),
            ]
        ),
    )
)
models.append(
    (
        "DTREE",
        Pipeline(
            steps=[
                ("scaler", StandardScaler()),
                ("decision_tree", DecisionTreeClassifier(random_state=1)),
            ]
        ),
    )
)

results = []  # Empty list to store all model's CV scores
names = []  # Empty list to store name of the models

# loop through all models to get the mean cross validated score
for name, model in models:
    scoring = "recall"
    kfold = StratifiedKFold(
        n_splits=5, shuffle=True, random_state=1
    )  # Setting number of splits equal to 5
    cv_result = cross_val_score(
        estimator=model, X=X_train, y=y_train, scoring=scoring, cv=kfold
    )
    results.append(cv_result)
    names.append(name)
    print("{}: {}".format(name, cv_result.mean() * 100))
```

```
LR: 62.15781745111679
RF: 76.99667671381096
GBM: 84.81142283020326
ADB: 83.14282402040342
XGB: 88.67532266790323
DTREE: 79.98376999768142
```

In [619…

```
# Plotting boxplots for CV scores of all models defined above
fig = plt.figure(figsize=(10, 7))

fig.suptitle("Algorithm Comparison")
ax = fig.add_subplot(111)

plt.boxplot(results)
ax.set_xticklabels(names)

plt.show()
```



- We can see that XGBoost is giving the highest cross-validated recall followed by GradientBoost and AdaBoost
- The boxplot shows that the performance of all the models is consistent with no outlier.
- We will tune 3 models - XGBoost , GradientBoost , AdaBoost and see if the performance improves.

Hyperparameter Tuning

- We will use pipelines with StandardScaler and XGBoost model and tune the model using GridSearchCV and RandomizedSearchCV. We will also compare the performance and time taken by these two methods - grid search and randomized search.**
- We can also use the make_pipeline function instead of Pipeline to create a pipeline.**
- "make_pipeline": This is a shorthand for the Pipeline constructor; it does not require and does not permit, naming the estimators. Instead, their names will be set to the lowercase of their types automatically.

In [620…

```
##  Function to calculate different metric scores of the model - Accuracy, Recall and P
def get_metrics_score(model, flag=True):
    """
    model: classifier to predict values of X

    """
```

```
    # defining an empty list to store train and test results
    score_list = []

    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)

    train_acc = model.score(X_train, y_train)
    test_acc = model.score(X_test, y_test)

    train_recall = metrics.recall_score(y_train, pred_train)
    test_recall = metrics.recall_score(y_test, pred_test)

    train_precision = metrics.precision_score(y_train, pred_train)
    test_precision = metrics.precision_score(y_test, pred_test)

    score_list.extend(
        (
            train_acc,
            test_acc,
            train_recall,
            test_recall,
            train_precision,
            test_precision,
        )
    )

    # If the flag is set to True then only the following print statements will be dispa
    if flag == True:
        print("Accuracy on training set : ", model.score(X_train, y_train))
        print("Accuracy on test set : ", model.score(X_test, y_test))
        print("Recall on training set : ", metrics.recall_score(y_train, pred_train))
        print("Recall on test set : ", metrics.recall_score(y_test, pred_test))
        print(
            "Precision on training set : ", metrics.precision_score(y_train, pred_train
        )
        print("Precision on test set : ", metrics.precision_score(y_test, pred_test))

    return score_list  # returning the list with train and test scores
```

In [621…

```
## Function to create confusion matrix
def make_confusion_matrix(model, y_actual, labels=[1, 0]):
    """
    model: classifier to predict values of X
    y_actual: ground truth

    """
    y_predict = model.predict(X_test)
    cm = metrics.confusion_matrix(y_actual, y_predict, labels=[0, 1])
    df_cm = pd.DataFrame(
        cm,
        index=[i for i in ["Actual - No", "Actual - Yes"]],
        columns=[i for i in ["Predicted - No", "Predicted - Yes"]],
    )
    group_counts = ["{0:0.0f}".format(value) for value in cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in cm.flatten() / np.sum(cm)
    labels = [f"{v1}\n{v2}" for v1, v2 in zip(group_counts, group_percentages)]
    labels = np.asarray(labels).reshape(2, 2)
    plt.figure(figsize=(10, 7))
    sns.heatmap(df_cm, annot=labels, fmt="")
```

```
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

XGBoost

GridSearchCV

In [622…

```
%%time

#Creating pipeline
pipe=make_pipeline(StandardScaler(), XGBClassifier(random_state=1,eval_metric='logloss'

#Parameter grid to pass in GridSearchCV
param_grid={'xgbclassifier__n_estimators':np.arange(10,100,30),
            'xgbclassifier__scale_pos_weight':[0,1,2,5,10],
            'xgbclassifier__learning_rate':[0.01,0.1,0.2,0.05],
            'xgbclassifier__gamma':[0,1,3],
            'xgbclassifier__subsample':[0.7,0.8,0.9,1]}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling GridSearchCV
grid_cv = GridSearchCV(estimator=pipe, param_grid=param_grid, scoring=scorer, cv=5, n_j

#Fitting parameters in GridSeachCV
grid_cv.fit(X_train,y_train)


print("Best parameters are {} with CV score={}:" .format(grid_cv.best_params_,grid_cv.b
```

```
Best parameters are {'xgbclassifier__gamma': 3, 'xgbclassifier__learning_rate': 0.05, 'x
gbclassifier__n_estimators': 70, 'xgbclassifier__scale_pos_weight': 10, 'xgbclassifier__
subsample': 0.7} with CV score=0.9473220496174356:
Wall time: 9min 2s
```

In [623…

```
# Creating new pipeline with best parameters
xgb_tuned1 = make_pipeline(
    StandardScaler(),
    XGBClassifier(
        random_state=1,
        n_estimators=70,
        scale_pos_weight=10,
        subsample=0.7,
        learning_rate=0.05,
        gamma=3,
        eval_metric='logloss',
    ),
)

# Fit the model on training data
xgb_tuned1.fit(X_train, y_train)
```

Out[623…

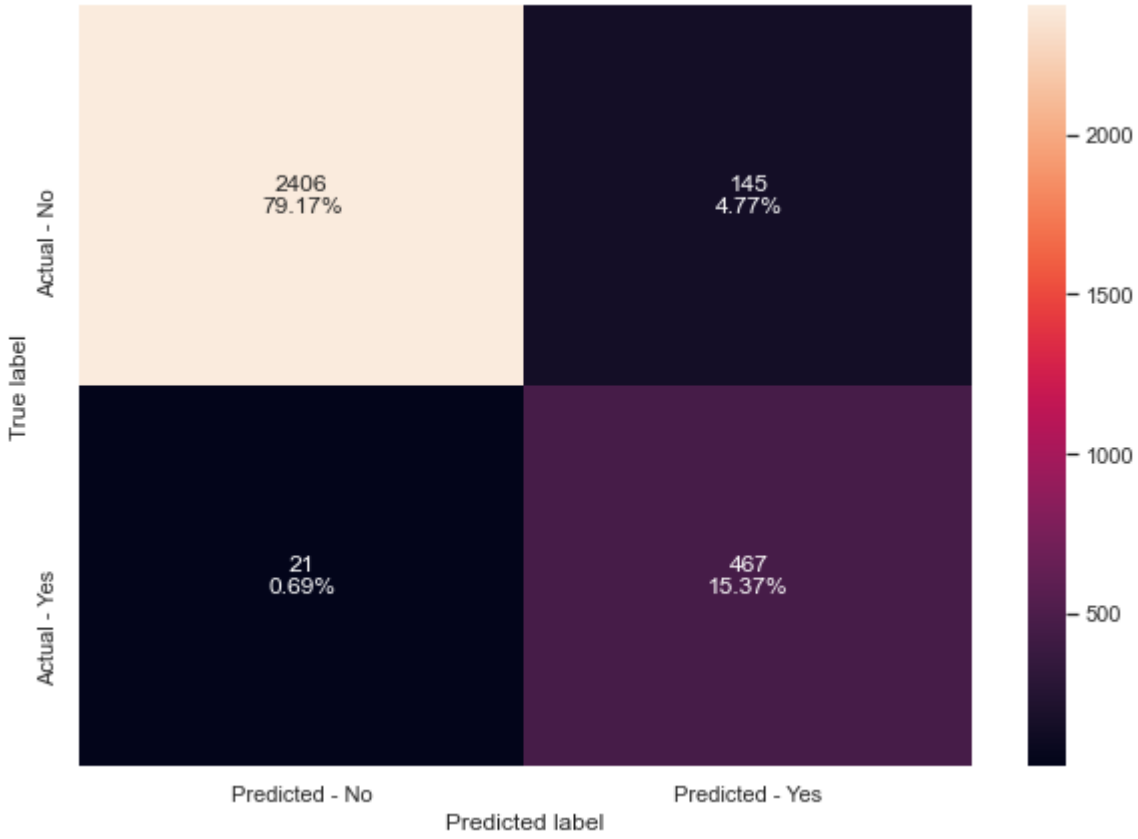
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('xgbclassifier',
                 XGBClassifier(base_score=0.5, booster='gbtree',
                               colsample_bylevel=1, colsample_bynode=1,
                               colsample_bytree=1, eval_metric='logloss',
                               gamma=3, gpu_id=-1, importance_type='gain',
                               interaction_constraints='', learning_rate=0.05,
```

```
                                max_delta_step=0, max_depth=6,
                                min_child_weight=1, missing=nan,
                                monotone_constraints='()', n_estimators=70,
                                n_jobs=4, num_parallel_tree=1, random_state=1,
                                reg_alpha=0, reg_lambda=1, scale_pos_weight=10,
                                subsample=0.7, tree_method='exact',
                                validate_parameters=1, verbosity=None))])
```

In [624…

```
# Calculating different metrics
get_metrics_score(xgb_tuned1)

# Creating confusion matrix
make_confusion_matrix(xgb_tuned1, y_test)
```

```
Accuracy on training set :  0.9672686230248307
Accuracy on test set :  0.9453767686739059
Recall on training set :  0.9991220368744512
Recall on test set :  0.9569672131147541
Precision on training set :  0.831263696128561
Precision on test set :  0.7630718954248366
```



- The test recall has increased by ~10% as compared to the result from cross-validation with default parameters.
- The model is overfitting the training data.

RandomizedSearchCV

In [625…

```
%%time

#Creating pipeline
pipe=make_pipeline(StandardScaler(),XGBClassifier(random_state=1,eval_metric='logloss',

param_grid={'xgbclassifier__n_estimators':np.arange(10,100,30),
```

```
            'xgbclassifier__scale_pos_weight':[0,1,2,5,10],
            'xgbclassifier__learning_rate':[0.01,0.1,0.2,0.05],
            'xgbclassifier__gamma':[0,1,3],
            'xgbclassifier__subsample':[0.7,0.8,0.9,1]}


# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=pipe, param_distributions=param_grid, n_it

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train,y_train)

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,ran
```

```
Best parameters are {'xgbclassifier__subsample': 0.8, 'xgbclassifier__scale_pos_weight':
10, 'xgbclassifier__n_estimators': 40, 'xgbclassifier__learning_rate': 0.2, 'xgbclassifi
er__gamma': 3} with CV score=0.9420588917226989:
Wall time: 58 s
```

In [626…

```
# Creating new pipeline with best parameters
xgb_tuned2 = Pipeline(
    steps=[
        ("scaler", StandardScaler()),
        (
            "XGB",
            XGBClassifier(
                random_state=1,
                n_estimators=40,
                scale_pos_weight=10,
                gamma=3,
                subsample=0.8,
                learning_rate= 0.2,
                eval_metric='logloss', max_depth = 2, reg_lambda = 2
            ),
        ),
    ]
)
# Fit the model on training data
xgb_tuned2.fit(X_train, y_train)
```

Out[626…

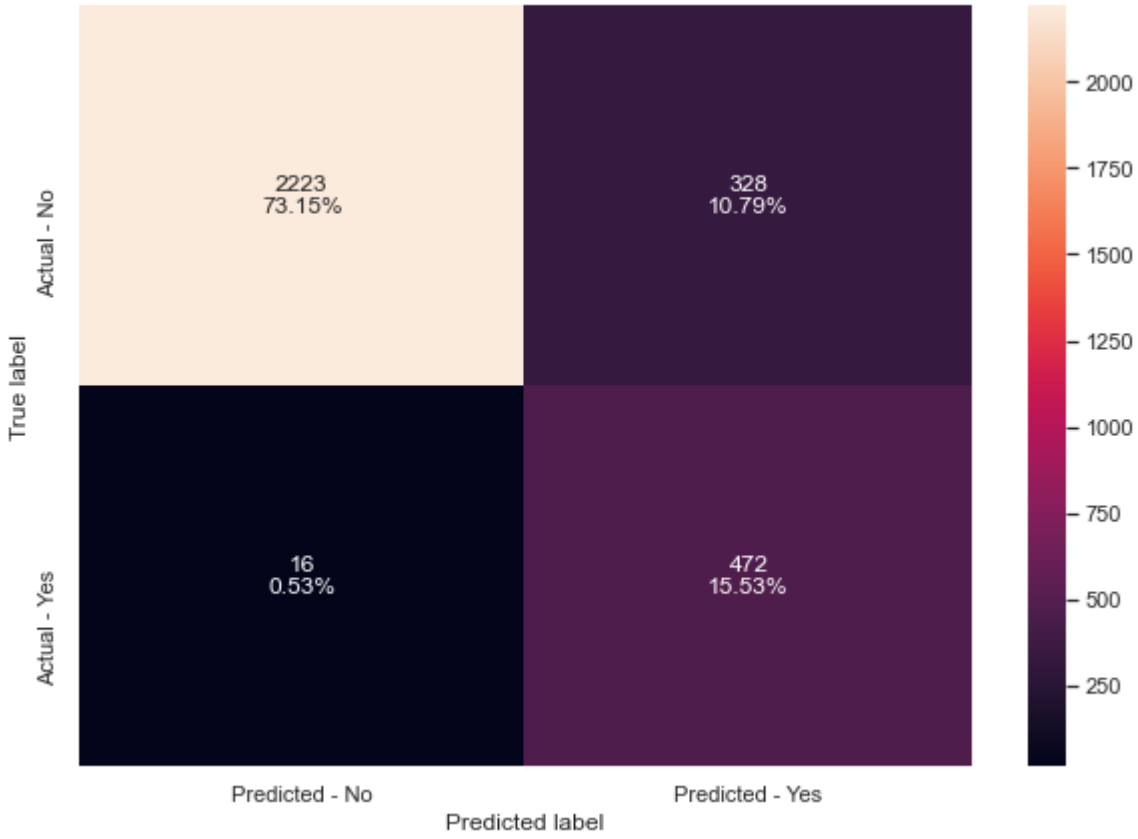
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('XGB',
                 XGBClassifier(base_score=0.5, booster='gbtree',
                               colsample_bylevel=1, colsample_bynode=1,
                               colsample_bytree=1, eval_metric='logloss',
                               gamma=3, gpu_id=-1, importance_type='gain',
                               interaction_constraints='', learning_rate=0.2,
                               max_delta_step=0, max_depth=2,
                               min_child_weight=1, missing=nan,
                               monotone_constraints='()', n_estimators=40,
                               n_jobs=4, num_parallel_tree=1, random_state=1,
                               reg_alpha=0, reg_lambda=2, scale_pos_weight=10,
                               subsample=0.8, tree_method='exact',
                               validate_parameters=1, verbosity=None))])
```

In [627…

```
# Calculating different metrics
get_metrics_score(xgb_tuned2)
```

```
# Creating confusion matrix
make_confusion_matrix(xgb_tuned2, y_test)
```

```
Accuracy on training set :  0.897714446952596
Accuracy on test set :  0.8868048700230339
Recall on training set :  0.9859525899912204
Recall on test set :  0.9672131147540983
Precision on training set :  0.6129912663755459
Precision on test set :  0.59
```



- Random search is giving better results than Grid search.
- The test recall has increased as compared to the test recall from grid search but the accuracy and precision have decreased.

GradientBoost

GridSearchCV

In [628…

```
%%time

# Creating pipeline
pipe = make_pipeline(StandardScaler(), GradientBoostingClassifier(random_state=1))

param_grid = {
    "gradientboostingclassifier__n_estimators": [100,150,200],
    "gradientboostingclassifier__subsample":[0.8,0.9,1],
    "gradientboostingclassifier__max_features":[0.7,0.8,0.9,1]
}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)
```

```
# Calling GridSearchCV
grid_cv = GridSearchCV(estimator=pipe, param_grid=param_grid, scoring=scorer, cv=5, n_j

# Fitting parameters in GridSeachCV
grid_cv.fit(X_train, y_train)

print(
    "Best Parameters:{} \nScore: {}".format(grid_cv.best_params_, grid_cv.best_score_)
)
```

```
Best Parameters:{'gradientboostingclassifier__max_features': 0.9, 'gradientboostingclass
ifier__n_estimators': 200, 'gradientboostingclassifier__subsample': 0.8}
Score: 0.8788391684055956
Wall time: 1min 22s
```

In [629…

```
# Creating new pipeline with best parameters
gbc_tuned1 = make_pipeline(
    StandardScaler(),
    GradientBoostingClassifier(
        n_estimators=200,
        max_features=0.9,
        subsample=0.8,
        random_state=1,
    ),
)

# Fit the model on training data
gbc_tuned1.fit(X_train, y_train)
```

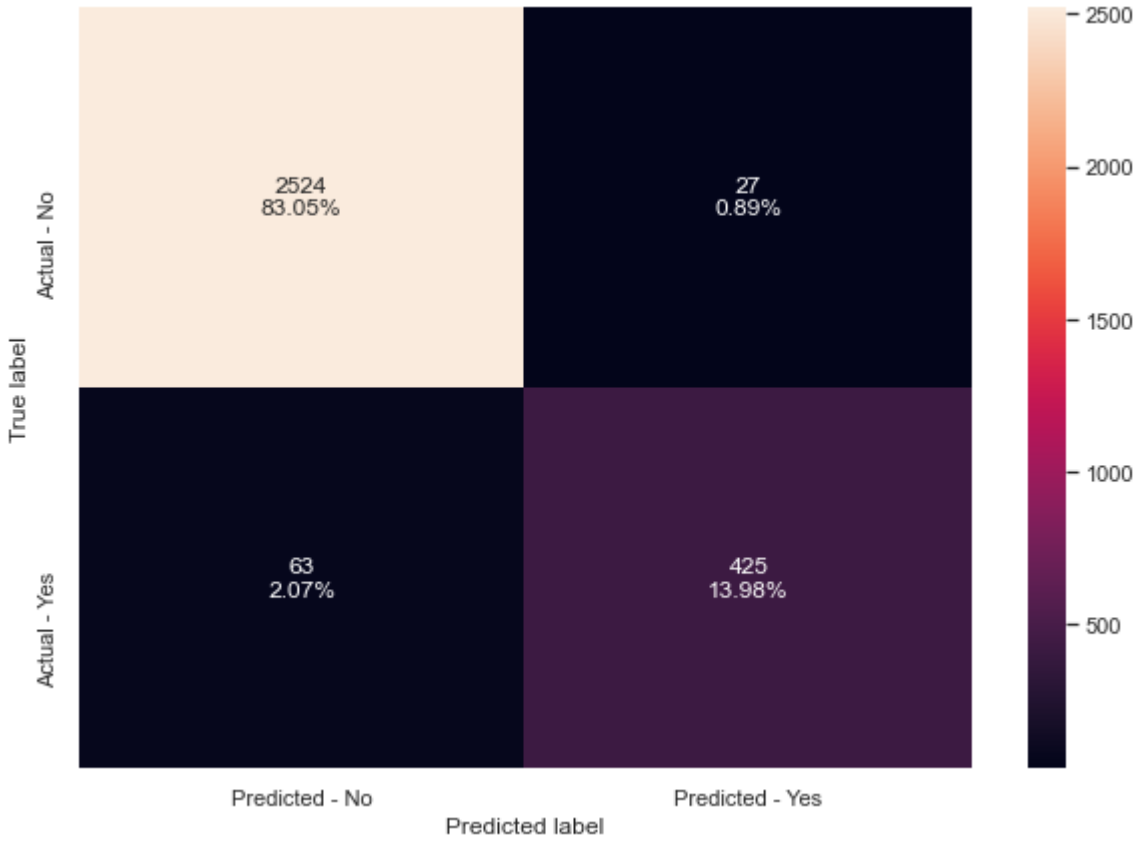
Out[629…

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('gradientboostingclassifier',
                 GradientBoostingClassifier(max_features=0.9, n_estimators=200,
                                            random_state=1, subsample=0.8))])
```

In [630…

```
# Calculating different metrics
get_metrics_score(gbc_tuned1)

# Creating confusion matrix
make_confusion_matrix(gbc_tuned1, y_test)
```

```
Accuracy on training set :  0.9874435665914221
Accuracy on test set :  0.9703849950641659
Recall on training set :  0.9464442493415277
Recall on test set :  0.8709016393442623
Precision on training set :  0.9746835443037974
Precision on test set :  0.9402654867256637
```



- The test recall has increased by ~3% as compared to cross-validated recall
- The tuned Gradientboost model is slightly overfitting the training data

RandomizedSearchCV

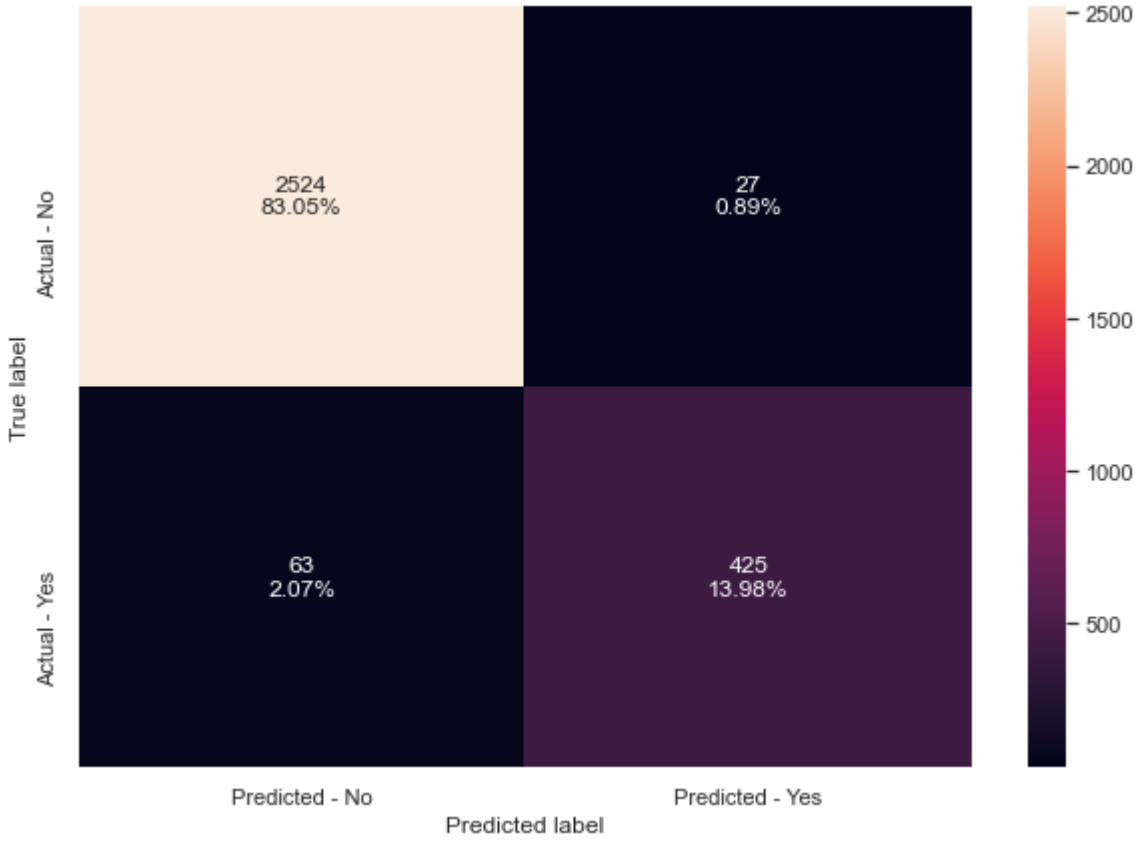
In [631…

```
%%time

# Creating pipeline
pipe = make_pipeline(StandardScaler(), GradientBoostingClassifier(random_state=1))

# Parameter grid to pass in RandomizedSearchCV
param_grid = {
    "gradientboostingclassifier__n_estimators": [100,150,200],
    "gradientboostingclassifier__subsample":[0.8,0.9,1],
    "gradientboostingclassifier__max_features":[0.7,0.8,0.9,1]
}
# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
gbc_tuned2 = RandomizedSearchCV(estimator=pipe, param_distributions=param_grid, n_iter=

#Fitting parameters in RandomizedSearchCV
gbc_tuned2.fit(X_train,y_train)

print("Best parameters are {} with CV score={}:" .format(gbc_tuned2.best_params_,gbc_tu
```

```
Best parameters are {'gradientboostingclassifier__subsample': 0.8, 'gradientboostingclas
sifier__n_estimators': 200, 'gradientboostingclassifier__max_features': 0.9} with CV sco
re=0.8788391684055956:
Wall time: 4min 1s
```

In [632…
```
# Calculating different metrics
get_metrics_score(gbc_tuned2)

# Creating confusion matrix
make_confusion_matrix(gbc_tuned2, y_test)
```

```
Accuracy on training set :  0.9464442493415277
Accuracy on test set :  0.8709016393442623
Recall on training set :  0.9464442493415277
Recall on test set :  0.8709016393442623
Precision on training set :  0.9746835443037974
Precision on test set :  0.9402654867256637
```



AdaBoost

GridSearchCV

In [633…
```
%%time

# Creating pipeline
pipe = make_pipeline(StandardScaler(), AdaBoostClassifier(random_state=1))

# Parameter grid to pass in GridSearchCV
param_grid = {
    "adaboostclassifier__n_estimators": np.arange(10, 110, 10),
    "adaboostclassifier__learning_rate": [0.1, 0.01, 0.2, 0.05, 1],
    "adaboostclassifier__base_estimator": [
        DecisionTreeClassifier(max_depth=1, random_state=1),
        DecisionTreeClassifier(max_depth=2, random_state=1),
        DecisionTreeClassifier(max_depth=3, random_state=1),
    ],
}
```

```
# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

# Calling GridSearchCV
grid_cv = GridSearchCV(estimator=pipe, param_grid=param_grid, scoring=scorer, cv=5, n_j

# Fitting parameters in GridSeachCV
grid_cv.fit(X_train, y_train)

print(
    "Best Parameters:{} \nScore: {}".format(grid_cv.best_params_, grid_cv.best_score_)
)
```

```
Best Parameters:{'adaboostclassifier__base_estimator': DecisionTreeClassifier(max_depth=
2, random_state=1), 'adaboostclassifier__learning_rate': 1, 'adaboostclassifier__n_estim
ators': 80}
Score: 0.8858683051240435
Wall time: 2min 46s
```

In [634…

```
# Creating new pipeline with best parameters
abc_tuned1 = make_pipeline(
    StandardScaler(),
    AdaBoostClassifier(
        base_estimator=DecisionTreeClassifier(max_depth=2, random_state=1),
        n_estimators=80,
        learning_rate=1,
        random_state=1,
    ),
)

# Fit the model on training data
abc_tuned1.fit(X_train, y_train)
```

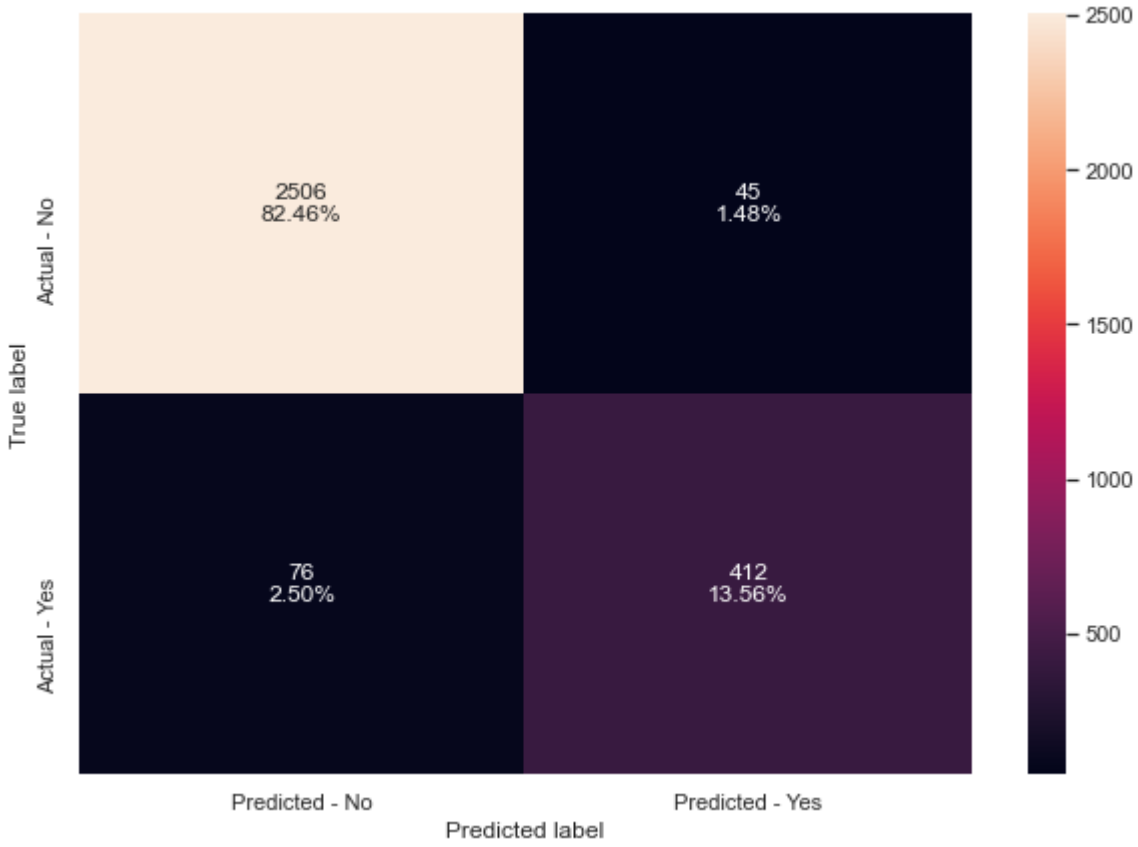
Out[634…

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('adaboostclassifier',
                 AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=2,
                                                                          random_state=
1),
                                    learning_rate=1, n_estimators=80,
                                    random_state=1))])
```

In [635…

```
# Calculating different metrics
get_metrics_score(abc_tuned1)

# Creating confusion matrix
make_confusion_matrix(abc_tuned1, y_test)
```

```
Accuracy on training set :  0.9922404063205418
Accuracy on test set :  0.9601842711418229
Recall on training set :  0.9701492537313433
Recall on test set :  0.8442622950819673
Precision on training set :  0.9813499111900533
Precision on test set :  0.9015317286652079
```



- The test recall has increased by ~3% as compared to cross-validated recall
- The tuned Adaboost model is slightly overfitting the training data

RandomizedSearchCV

In [636…

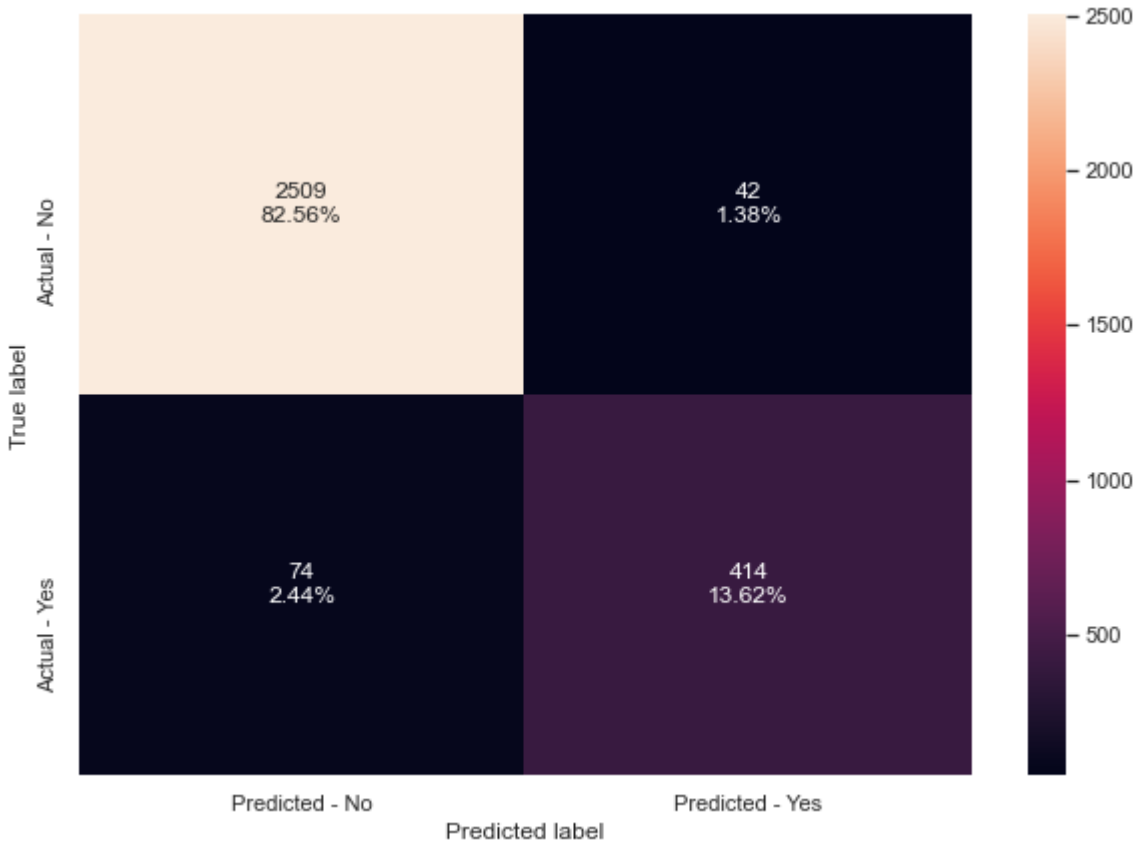
```
%%time

# Creating pipeline
pipe = make_pipeline(StandardScaler(), AdaBoostClassifier(random_state=1))

# Parameter grid to pass in RandomizedSearchCV
param_grid = {
    "adaboostclassifier__n_estimators": np.arange(10, 110, 10),
    "adaboostclassifier__learning_rate": [0.1, 0.01, 0.2, 0.05, 1],
    "adaboostclassifier__base_estimator": [
        DecisionTreeClassifier(max_depth=1, random_state=1),
        DecisionTreeClassifier(max_depth=2, random_state=1),
        DecisionTreeClassifier(max_depth=3, random_state=1),
    ],
}
# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
abc_tuned2 = RandomizedSearchCV(estimator=pipe, param_distributions=param_grid, n_iter=

#Fitting parameters in RandomizedSearchCV
abc_tuned2.fit(X_train,y_train)

print("Best parameters are {} with CV score={}:" .format(abc_tuned2.best_params_,abc_tu
```

```
Best parameters are {'adaboostclassifier__n_estimators': 90, 'adaboostclassifier__learni
ng_rate': 1, 'adaboostclassifier__base_estimator': DecisionTreeClassifier(max_depth=2, r
andom_state=1)} with CV score=0.8849833835690548:
Wall time: 3min
```

In [637…

```
# Calculating different metrics
get_metrics_score(abc_tuned2)

# Creating confusion matrix
make_confusion_matrix(abc_tuned2, y_test)
```

```
Accuracy on training set :  0.9789288849868305
Accuracy on test set :  0.8483606557377049
Recall on training set :  0.9789288849868305
Recall on test set :  0.8483606557377049
Precision on training set :  0.9858532272325375
Precision on test set :  0.9078947368421053
```



Comparing all models

In [638…

```
# defining list of models
models = [xgb_tuned1, xgb_tuned2, gbc_tuned1, gbc_tuned2, abc_tuned1, abc_tuned2]

# defining empty lists to add train and test results
acc_train = []
acc_test = []
recall_train = []
recall_test = []
precision_train = []
precision_test = []

# looping through all the models to get the metrics score - Accuracy, Recall and Precis
for model in models:
```

```
    j = get_metrics_score(model, False)
    acc_train.append(j[0])
    acc_test.append(j[1])
    recall_train.append(j[2])
    recall_test.append(j[3])
    precision_train.append(j[4])
    precision_test.append(j[5])
```

In [639…

```
comparison_frame = pd.DataFrame(
    {
        "Model": [
            "XGBoost with GridSearchCV",
            "XGBoost with RandomizedSearchCV",
            "GradientBoost with GridSearchCV",
            "GradientBoost with RandomizedSearchCV",
            "Adaboost with GridSearchCV",
            "Adaboost with RandomizedSearchCV",


        ],
        "Train_Accuracy": acc_train,
        "Test_Accuracy": acc_test,
        "Train_Recall": recall_train,
        "Test_Recall": recall_test,
        "Train_Precision": precision_train,
        "Test_Precision": precision_test,
    }
)

# Sorting models in decreasing order of test recall
comparison_frame.sort_values(by="Test_Recall", ascending=False)
```

Out[639…

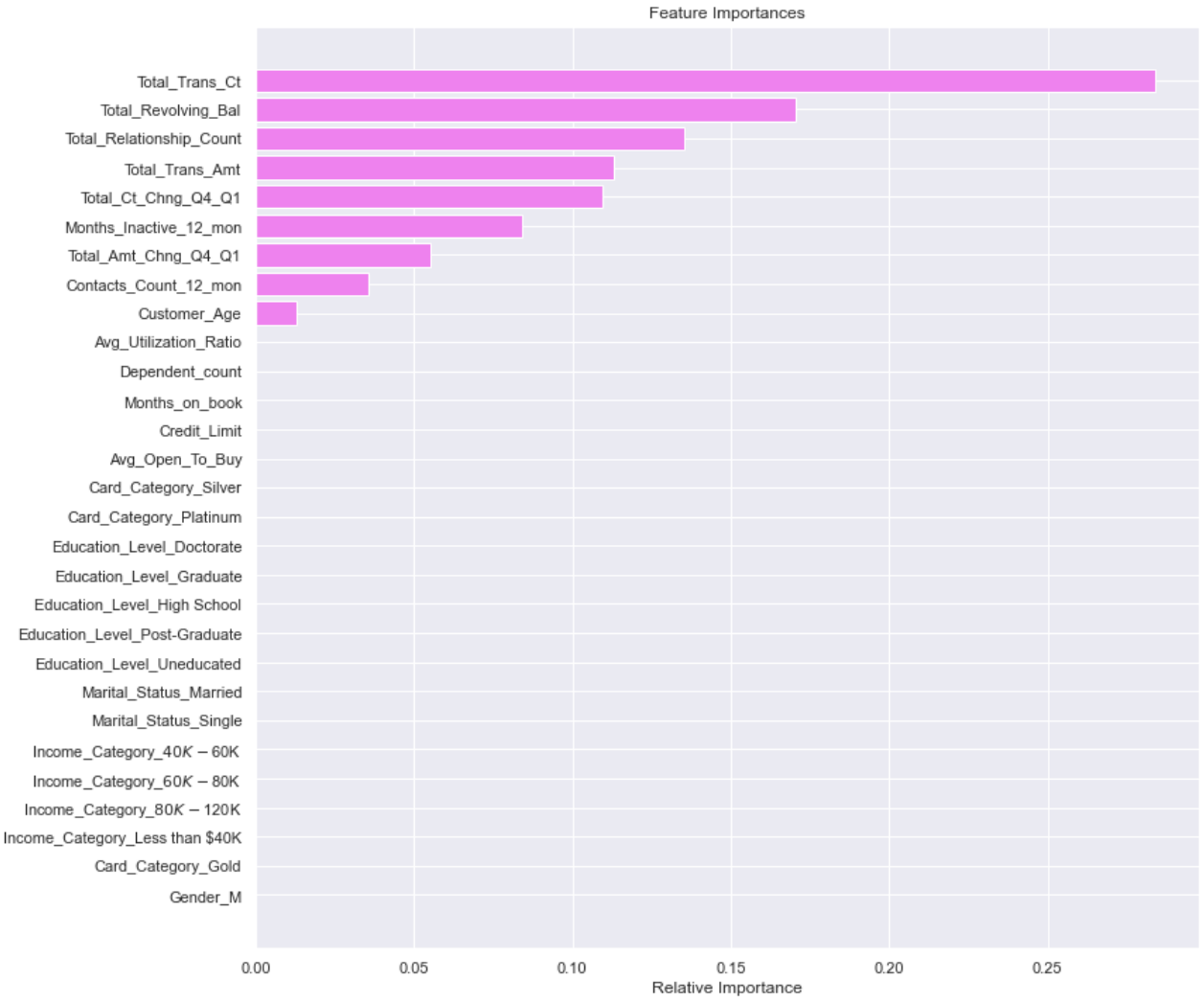
	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Pre
1	XGBoost with RandomizedSearchCV	0.897714	0.886805	0.985953	0.967213	0.612991	0.5
0	XGBoost with GridSearchCV	0.967269	0.945377	0.999122	0.956967	0.831264	0.7
2	GradientBoost with GridSearchCV	0.987444	0.970385	0.946444	0.870902	0.974684	0.9
3	GradientBoost with RandomizedSearchCV	0.946444	0.870902	0.946444	0.870902	0.974684	0.9
5	Adaboost with RandomizedSearchCV	0.978929	0.848361	0.978929	0.848361	0.985853	0.9
4	Adaboost with GridSearchCV	0.992240	0.960184	0.970149	0.844262	0.981350	0.9

- The `xgboost model tuned using randomised search` is giving the best test recall of 0.96 but it has the least train and test precision.
- Compared to `Logistic Regression on Undersampled data` with 0.811 Test_Recall, `xgboost model tuned using randomised search` is the best model performer.

Feature importance from the tuned xgboost model

In [640…

```
feature_names = X_train.columns
importances = xgb_tuned2[1].feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12, 12))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



- `Total_Trans_Ct` is the most important feature, followed by `Total_Revolving_Bal` and `Total_Relationship_Count` of the customer.

Business Recommendation

- Company should target customers with low Total_Trans_Ct -Total Transaction Count (Last 12 months) [Total_Trans_Ct value 50 and lower]
- Company should target customers with low Total_Trans_Amt -Total Transaction Amount (Last 12 months) [Total_Trans_Amt $3000 and lower]

- Company should target customers with low Total_Revolving_Bal: The balance that carries over from one month to the next is the revolving balance [Total_Revolving_Bal value $1400 and lower]
- Company should target customers with low Total_Relationship_Count: Total no. of products held by the customer [Customers with Total_Relationship_Count values 1,2,3 should be targeted.]

In [ ]:

End-of-File